Math

9

UIUCDCS-R-73-604

COO-2118-0049

PICTURE ANALYSIS BY GRAPH TRANSFORMATION

by

Ahmad E. Masumi

October 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-604

PICTURE ANALYSIS BY GRAPH TRANSFORMATION

by

Ahmad E. Masumi

October 1973

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Supported in part by the Department of Computer Science and the Atomic Energy Commission under contract US AEC AT(11-1)2118 and submitted in partial fulfillment of the requirements of the Graduate College for the degree of Doctor of Philosphy in Computer Science.

ACKNOWLEDGMENT

I extend my deepest gratitude to my thesis advisor, Professor Bruce H. McCormick, who in spite of his departure from the Department of Computer Science and heavy new responsibilities as the head of Information Engineering Department of the University of Illinois at Chicago Circle, has been the prime contributor to the completion of this thesis.

I also want to extend my cordial regards to Professor J. N. Snyder, the head of our department, for his understanding and provision of necessary funds for this research.

My best regards are also extended to Mrs. Judy Rudicil who has been extremely patient with my hand writing and has typed a fine thesis. Many thanks to Stanley Zundo who raced with time to finish the figures and drawings on time.

TABLE OF CONTENTS

1. INTRODUCTION

Our long range goal is to develop a system which is able to analyze its visual environment--in other words, a system which is able to see--and functions intelligently based on the extracted information from the scenes.

Our immediate objective in this work is to establish and identify the integral parts of an artificial system capable of visual perception. We mainly concern ourselves with global aspects of processing, i.e. some pre-processing is assumed on the picture before it is treated by this model. These processes typically consist of operations which can abstract low-level primitive elements as well as find relations between the elements of the scene.

The deduction is accomplished by a graph structure. Due to the fact that we cannot describe a picture in terms of strings of subpictures (but for a few exceptional classes of pictures), phrase-structure grammars cannot be used directly. The rewriting rules must act on more general entities such as arrays, drawings, labeled graphs (webs), multigraphs, etc. For example, Kirsch (1964) and Dacey (1967) designed a grammar for two-dimensional languages where the generating rules act on arrays. Pflatz and Rosenfeld (1969) used for picture description the so-called web grammars in which the rules act on labeled directed graphs. Simply, in a picture grammar one tries to replace the rigid ordering of symbols by partial ordering of graph structure so that the parsing can still work.

1.1 Relevant Background

A 1966 collection of papers edited by Leonard Uhr [1] deals with various problems of pattern recognition by computers. However, many of the more crucial problems such as how humans encode shape information detect near

similarity or dissimilarity of shapes, focus attention upon particular aspects of shapes, extract global information from acquired local information, and achieve perceptual invariance are not yet adequately understood.

In processing pictures with the help of computers, the problem that has been most extensively studied is the recognition problem. Typically, this has posed as a problem in categorization as follows: given a finite set of picture prototypes and a token of one of the prototypes, the task is to assign the token to the correct prototype. Attempts to solve this problem have traditionally been decision-theoretic in their approaches. With each prototype is associated a list of attributes and with each attribute a set of values. The space of attribute values is then partitioned into mutually disjoint regions and each region is assigned to one of the prototypes. Given a token now, its attribute values are computed and using these computed values one determines to which region in the property space it belongs. Accordingly, the input token is categorized as belonging to one or another prototype.

Michalski [2] defines a variable valued logic system which is capable of this kind of categorization. In this approach, the attributes must have discrete values or the range of values divided to discrete intervals. This procedure treats each prototype and token as a single atomic entity. The methodology of attribute assignments and attribute value computations is guided primarily by the desire to optimize the partitioning of the property space and to devise inference techniques to make minimal error decisions on the basis of statistics computed. No other aspects of picture description or analysis play any role in this approach.

A more careful study and critical analysis of earlier works would show certain inherent inadequacies in this general methodology. These

inadequacies relate to the fact that in dealing with pictures (or, more properly, classes of pictures) the really relevant and significant problems are concerned with picture description and its analysis. Picture recognition is actually only one aspect of this larger problem of picture analysis. Hence, an adequate framework for coping with the recognition problem in its generality must be capable of analyzing the input picture and generating a structured description of it, and not be restricted merely to making a "YES," "NO," "DONT KNOW" decision.

We emphasize that the argument here is not that the classificatory schemata and descriptive schemata are mutually exclusive, or otherwise incompatible, in solving the recognition problem. Rather, our approach will positively incorporate this classificatory information in a recognition technique based on the structural description of the input picture. A classificatory technique based on property lists is a degenerate description with the structural information missing.

Abstraction of information is an essential part of the activities that any intelligent system would have to perform in the course of analysis and recognition of pictures. A picture may be regarded as a function over some two-dimensional domain. By appropriate choice of sampling grid size and quantization levels, any picture function can be regarded as indistinguishable from a m x n array with elements p_{ij} in some bounded range $[0, 2^{k}_{-1}]$. This array is a "faithful description" of the original scene in the sense that both have virtually the same information content. Picture processing research is largely concerned with the effective transmission and analysis of these digital descriptions. In many contexts, however, it is preferred to have a description that is not faithful, but nevertheless reflects the "essential" information in the scene relative to some problem context.

Among this category is the work of Maruyama [3], where he represents several region finding algorithms, and several shape representation techniques which easily lend themselves to feature extraction. This abstracted information is in turn used in the global analysis. In [4] Jayaramamurthy describes different methods of describing and analyzing textures, and in a recent article Rosenfeld [5] described how pictures could be divided into subregions through texture analysis.

Based on these and other successful attempts in low level picture processing, it is apparent that we need a global picture analyzer which will try to use all local information and features present in the scene.

Fundamental to the development of higher level picture processing procedures is the creation of a suitable picture representation for algorithms and data. This representation should express the hierarchical structures of elements with attributes and relations among them. Basic to this model is a graph-structured data representation and graph transformational passing procedures, which are central issues in our work. The graph structural representation model has the following features which facilitates flexibility:

(1) Attribute values associated with each element may be used to tie down the model to concrete instances.

(2) Interaction or propagation of information between parsing levels, which is needed to identify objects in context, is readily expressible.

Context sensitivity in pictures is best expressed by Guzman in [6] as follows:

> "Given that a set is formed by components that locally (by their shape, for example) are ambiguous, because they can have one of several values (O = sun, ball, eye, hole, etc.) or meanings, can we make use of <u>context information</u> ( occurs often) stated in the form of

models in order to assign to each component a value that the whole set is consistent or makes global sense?"

Indeed, as we have found out, this context sensitivity plays a central role in parsing pictures. At any level in parsing, although the pieces match locally, the global match can be rejected because of contextual differences (relations). For example, if a composite object containing constituents A and B is to be formed; it will be the case that additional relations (constraints) between A and B are to be met.

1.2 Formulation of the Problem

In this thesis, the scene is represented as a graph, where nodes correspond to the primitive regions of the scene and branches are the existing relations between these regions. A descriptive pattern analysis attempts to build description of patterns. We refer to the procedures which form the core of this analysis as parsing procedures. The central problems attacked by these procedures are the location, isolation, and identification of objects in a picture. The current lack of computational sophistication in attacking these problems is attested to by elementary (by human standards) image processing that systems today are able to perform.

This graph representation of scenes facilitates recognition procedures based on graph transformations. The recognition process is viewed as the application of proper replacement rules to the graphical representation of the scene. In addition the imposition of a partial ordering on the graphical model of the objects known to the recognizer, facilitates the automatic inference of these replacement rules.

Other operations like creating a branch or merging two nodes, which may be dictated by the current picture segmentation strategy, etc., are required

in search for possible domains to apply graph transformations inferred by graph structural model system.

The development of a suitable processing language has been a necessary part of the solution to this problem. Currently adequate languages exist to express algorithms which operate on the array representation of pictures as binary valued elements with neighborhood connectivity relationships. The next and most natural abstraction from the array representation is a direct graph structure representation, by means of which many scene segmentation algorithms can be simply expressed. This language must have the operations necessary to implement structure transformations.

A structure operational language has been our choice for precisely specifying and experimenting with heuristic picture processing strategies in this research.

Languages to implement picture processing algorithms are divided into two categories; descriptive graphical languages and graph structure processing languages. Descriptive graphical languages have tended to be display-oriented and limited use in recognition of pictures. In this class are the systems and languages of Herzog [7], Kulsrud and Williams [8]. In Schwebel [9], some criteria for graphic languages are presented and a language, ICON, to meet these criteria is defined. Graph structure processing languages may be distinguished by the presence of operations which allow analysis of descriptions. Chase [10] uses a system for graph manipulation. Graph structures of greater generality can be treated with languages given in Pratt and Friedman [11], Early [12], Lieberman [13], Wolfberg [14], and Crespi-Reghiggi and Marpugo [15]. Webs have been used by J. L. Pflatz [16] in describing some classes of pictures, which have made the recognition of global patterns feasible.

The following requirements which are essential to any general graph-structural operation language have led us to select SOL, which has been originally proposed by J. C. Schwebel [17]. These requirements are listed in two parts: first the elements of the structure and secondly the requirements of the elements necessitated by the dynamic nature of the processing.

Static Requirements

Basic elements

nodes (picture primitives or higher level objects),

branches (relations between pairs of nodes),

graphs (collection of nodes and branches),

node attributes and their values,

branch attributes and their values,

graph attributes and their values.

Sets of elements

arbitrary sets of nodes and branches,

arbitrary sets of graphs,

graph levels (explicit substructures),

pointers to elements,

attributes and values for sets of elements.

Dynamic Requirements

add-delete operations,

functions on attribute values,

pointer move operations,

node operations,

set operations (Boolean),

structure replacement operations,

NAME and TYPE functions,

TAIL and HEAD functions,

subgraph operations.

The SOL language with some modification at conceptual level has been implemented. The language is implemented by embedding it in Pℓ/1 which is basically a procedural language. There are some inherent inefficiencies in list processing capability of Pℓ/1, but this language has been proved excellent for our demonstrative purposes.

The general organization of this thesis is as follows: in chapter two we introduce different existing methods of how pictures can be represented in a graphical form, and how the abstracted lower level information is propagated to this graph. Generally, the attribute values of graphical elements are passed to the graph by a set of feature extraction procedures, and are used as semantic classificatory information to guide the recognizer. We also lay the basic definition of a graph structure, which is the essential tool of representing the model of our universe.

In chapter three the theoretical ground rules of graph transformations are discussed. Here, we show the factors which affect the choice of domain for transformations, and further by introducing a useful class of relations, we show the context sensitivity of these transformations, and provide rules that are to be satisfied for the feasibility of transformations. We also show that our suggested set of relations are closed under some useful operations. In [17] Schwebel has set rules for general binary relations, but his rules are too restrictive to be useful in actual scene analysis.

In chapter four we introduce the graph structural representation of the universe's model and body of knowledge. In our model, the rules represent the skeleton of the objects to be encountered in our universe. In developing

the formal description of the system and parsing procedure, we also show the similarity with human perception and the psychological concept of attention. We have further discussed the use of associated attributes as heuristical means of speeding up the recognition process and propagation of knowledge from attention point to higher level objects. Here, the set of operations defined on the set of relations are extended to operate on the graph structure, and it is shown how a much larger class of objects can be recognized using these operations. The set of graphs (rules) in the graph structure form a partially ordered set and the immediate successors of any graph can also be ordered according to some criterion (frequency of occurrence, etc.). A few aspects of parallel processing are also discussed in this chapter.

In chapter five we have introduced our experimental universe. Graph structure representation of this universe and associated parsing procedures have been written in SOL. The computer results of this experimentation are tabulated and presented here in this chapter. The application of this general methodology of recognition to this simple class of pictures, namely "simplified cartoons in coloring books", has led us to the development of a set of useful concepts like best-match recognition of incomplete objects.

In chapter six the concept of learning is investigated and we will show that it is rather straightforward to incorporate it in our recognition system. The learning aspect of artificial intelligence is of extreme importance. "Learning" is a terminology which has been broadly discussed and disputed by many psychologists. We define "learning" as:

> "Ability to recognize the incomplete objects, introduce new objects to the universe, modify the conception of known objects, and enhance the performance of recognition through experience."

Many modern string language compilers have the capability of detecting and correcting the errors in the input stream. But, this is not considered to be learning capability, because this is just a fixed automation, and the performance would be the same from one run to another. Also it is difficult to introduce new patterns to the language grammar or modify the existing patterns to enable the system to accept broader class of patterns. These are incorporated in our system in the following manner.

The ability to recognize an incomplete object is tackled by the "best-match concept". That is, we can find the object which is closest in structure and semantics to the image and, if we are in "learning-mode" this information is saved in the "artificial brain."

The ability to introduce a new object is implemented by saving the structural and semantical information of the image as initial conception of this object, which is of course subject to modification through experiment of other instances of the same object.

As for modification of the conceptions, we reflect the conceived structural and semantical variations of the object in a temporary storage, and use some learning criterion to reflect these into permanent memory.

It is clear that this additional information will enhance the system's performance, both in enlarging the class of recognizable objects and in speeding up the recognition. We have used a scoring system as weighed branches between the objects in the graph structure which speeds up the recognition of more frequently occurring combinations.

In chapter seven we have explained the generality of our methodology. Although stiff restrictions have made our demonstration unattractive in immediate pragmatic applications, it is argued that once the other parts of the over all system, which are being extensively investigated or already

accomplished, are available this methodology will immediately emerge as a practical useful tool in artificial intelligence. We also discuss the other interesting features of this methodology, which we have barely touched upon in the course of this research. It is also my conviction that in long run experiments with this kind of system will lead to the creation of intelligent data bases which are relational in nature.

In Appendix A we have represented the definition of SOL and discuss its implementation.

In Appendix B a sample SOL program which represents an input scene image is given.

2. GRAPHICAL REPRESENTATION OF PICTURES

Our prime objective in this chapter is to demonstrate that the information present in a scene is best described in a graphical form, and further show the generality of region-node representation. J. L. Pfaltz in [16] states:

"A picture, photograph, or scene may be regarded as a function $p(x,y)$ over some two dimensional domain. This is a 'faithful description' of the original scene in the sense that both have virtually the same information content. Picture processing research is largely concerned with the effective transmission and analysis of these digital descriptions.

In many contexts, however, one would prefer a description that is not faithful, but that nevertheless reflects the 'essential' information in the scene relative to some problem context."

For example, the strings "an airplane is hidden in the woods," or "a boy is playing with a ball," may be far more adequate (and economical) descriptions than equivalent gray level description of digitized pictures. The implication is that in many actual situations there may be linguistic structures which

(1) are adequate descriptions of the pictures,

(2) present the "essential" information context of the picture in a more readily usable form,

(3) are more economical in terms of storage and transmission requirements, in that redundant information has been suppressed,

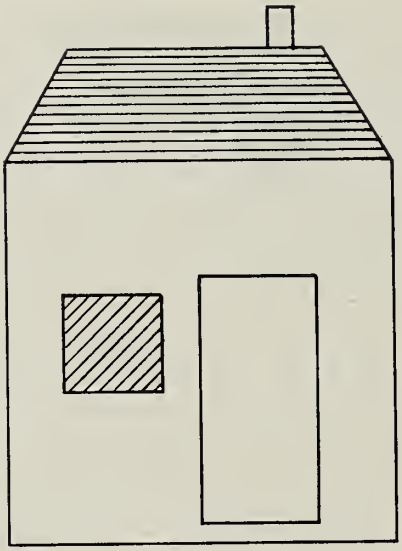
(4) have a built-in hierarchical structure,

(5) make the addition and deletion of objects easy.

We may formalize the idea of a picture description as follows: Let P denote a class of pictures. By a picture description language (PDL [18]) for P, we mean a set $\mathfrak{L}$ of linguistic structures such that for each $p \in P$ there exists an $L_p \in \mathfrak{L}$ called the description of p. Thus we have a picture description function (PDF), call it D, mapping P into $\mathfrak{L}$. In general there will be cases where $p_1, p_2 \in P$ are distinct, but $D(p_1) = D(p_2)$ in $\mathfrak{L}$. Differences between p_1 and p_2 in such a case are considered to be "noise" with respect to D. There are also cases where $L_p, L_p' \in \mathfrak{L}$ are distinct, but they are both descriptions of the same object. In this case, the language $\mathfrak{L}$ is said to be semantically ambiguous.

In the examples like "this is a picture of a house," the picture descriptions are all strings of symbols. This is natural, since humans usually use linear linguistic structures for communication. But the string-like structure of natural languages seem to be inappropriate, or at least inefficient, for general picture-description purposes. Consider, for example, the picture shown in Fig. 2.1 and its English description. This English language description is certainly not the only one possible, nor is it optimal in any sense; nevertheless, it may serve as an illustration. In particular, we notice that we have singled out and identified basic objects as "atoms," which are considered to be picture primitives; have identified certain properties of objects (size, shape, color, texture, etc.); have identified certain relations between objects (relative positions); and have indicated which objects have which properties, and which objects enter into which relationships.

In contrast to this English description, consider the linguistic structure of Fig. 2.2 as a description of the picture in Fig. 2.1. Readily, Fig. 2.2 conveys the same information about the picture, but it is in more



The picture consists of 2 squares, 2 rectangles, and 1 trapezoid; one of the squares has texture $\#T_1$, the other one has texture $\#T_0$. The big square "contains" the big rectangle and the small square. the trapezoid is directly above the big square; etc.

Fig. 2.1. A picture and its English description.

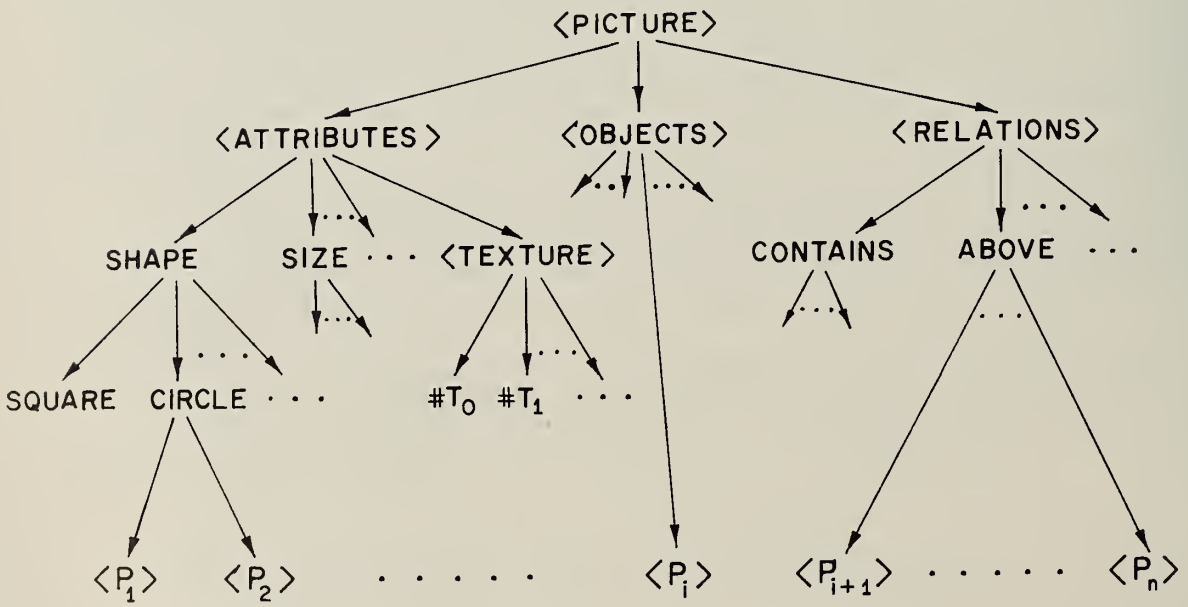


Fig. 2.2. A labeled graph description of Fig. 2.1.

usable form. It is immediately implementable as a linked data structure in which one can quickly determine which objects are square, what shape or size each is, etc. Of course the linguistic structure of Fig. 2.2 is merely a simple alternative to linear picture description languages, and was stated to show the existence of these alternatives.

2.1 Regions as Graph Nodes

We are convinced that the graphs are the best tools to describe the pictorial information; here we investigate the conversion of pictorial information into graphical structures. Graphs are widely used for the convenient representation of the geometric relations implicit in a picture consisting of many regions. By using a graph representation, many problems of picture analysis become feasible with provision of appropriate heuristic algorithms. The study of [19] Guzman (1968) demonstrated the significance of graphical representation of scenes of simple 3-dimensional shapes, using this representation to analyze and cluster planar regions into three-dimensional objects by extracting certain properties on the vertices. Eastman [20] (1970) finds a need for a better descriptive-capability to define the spatial relationships essential for two-dimensional space planning. Here again a graph representation is evoked.

It is obvious that there are many means to convert a picture to a graph, even with fixed criteria of what information should be preserved. The best way to arrive at our method of conversion is to investigate different alternatives of conversion of a simple line drawing to a graph and its proposed descriptional language.

Let us construct graphs for the picture of Fig. 2.3(a). Since each graph is essentially a collection of nodes and a set of branches among them,

the first question to be answered is, "What physical parts of the picture should be corresponded to the main abstract entities of the graph, namely nodes?" There are three possible answers to this question which we will investigate in turn.

Of the five criteria which we set as a judgment of good picture description, the first three are satisfied by having a graphical description language. So we have to base our judgment for selecting the best answer on the last two criteria.

2.1.1 Vertex-primitive

The first choice would be to have the geometrical topology of the graph directly correspond to that of the line drawing. Usually graphs are described in various matrix forms or in a sequential manner. None of these descriptions are flexible and powerful enough for our purposes. However, for illustrative purpose it would be useful to present a sequential representation of graph 2.3(b) similar to a method by Maruyama in [3]. Assume each node, i.e. intersection of more than two arcs, is denoted by the values of its x,y coordinates.

Let us assume the following expression for each node:

$$\langle V\rangle(\{\langle a_j\rangle(\langle R_i\rangle\langle V_k\rangle)\}^*) \ ,$$

where,

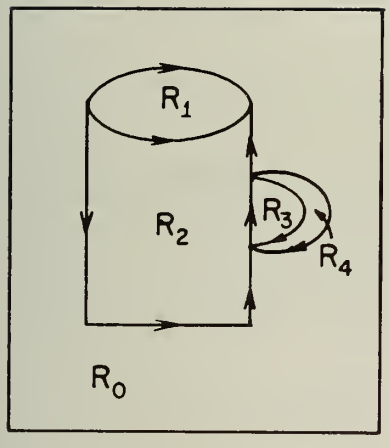
V:: = node name

V_k:: = node name different from V

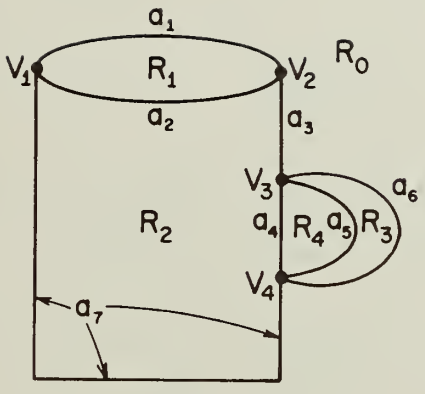
a_j:: = arc name

R_i:: = region name,

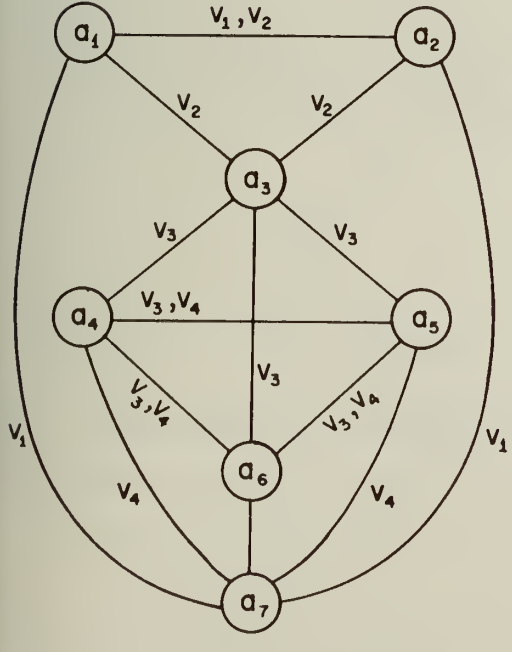
the following semantics are implied:



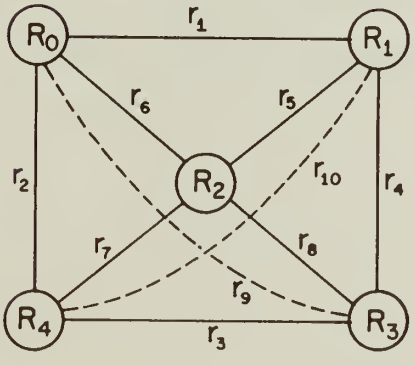
(a) picture of five regions



(b) vertex-node (naming)



(c) arc-node



(d) region-node

Fig. 2.3. Example of graph representation of a picture.

1) nodes V_k's are direct neighbors of the node V,

2) nodes V and V_k are connected with an arc a_j,

3) region R_i is (partly) bounded by arc a_j, and R_i is assumed by a_j in counter-clockwise direction at node V, and

4) the sequence of R_i's is cyclic.

By the application of the above representation, the picture of Fig. 2.3(a) can be described as follows:

$$V_1(a_1(R_0,V_2),a_7(R_2,V_4),a_2(R_1,V_2))$$

$$V_2(a_1(R_1,V_1),a_2(R_2,V_1),a_3(R_0,V_3))$$

$$V_3(a_3(R_2,V_2),a_6(R_3,V_4),a_5(R_4,V_4),a_6(R_0,V_4))$$

$$V_4(a_7(R_0,V_1),a_6(R_4,V_3),a_5(R_3,V_3),a_6(R_2,V_3)) \ .$$

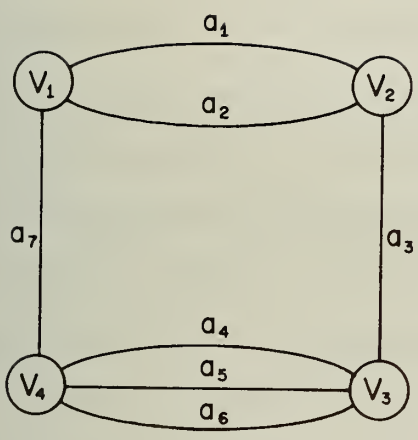
In the above expressions, arc a_j's are sequentially scanned in counter-clockwise order about node V. Thus the sequence of a_j's is cyclic. For example, the following two sequences are semantically identical:

$$V_1(a_1(R_0,V_2),a_7(R_2,V_4),a_2(R_1,V_2))$$

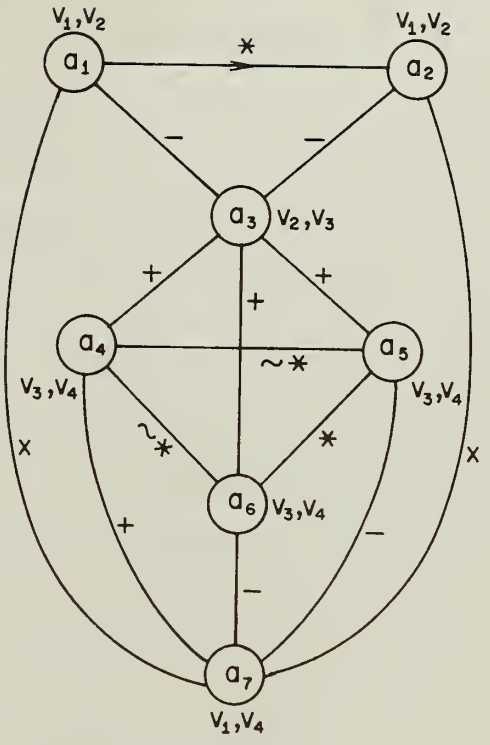
$$V_1(a_7(R_2,V_4),a_2(R_1,V_2),R_1(R_0,V_2)) \ .$$

As we observed in this case, sequential representations are bulky and often semantically ambiguous. In cases where arcs do not convey shape information and are simply straight lines, we can eliminate them from expressions. But still the most natural linguistic description would be the one which preserves the exact graphical structure of the picture, Fig. 2.4(a).

Lack of hierarchy and difficulty in adding or deleting parts to or from the picture makes this alternative very unattractive for our picture processing purposes.

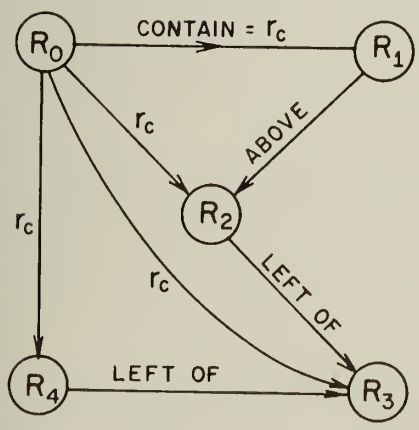


(a) vertex-node



(b) arc-node

{+-,x,*} are the same as in PDL.



(c) region-node

Fig. 2.4. Graphical descriptions of line drawing in Fig. 2.3(a).

2.1.2 Arc-primitive

The next alternative would be to have the arcs in the line drawing correspond to our picture primitive. In [18] Alan C. Shaw defines, "A picture primitive may be any n dimensional pattern with two distinguished points, a tail and a head. Primitives can be metrically concatenated together only at their tail and head points. Because the two points of possible concatenation are specified, a primitive can be represented as a labeled directed edge of a graph, pointing from its tail to head node." In this case the emphasis is still on having the picture graph and description graph correspond in geometrical topology. Based on this picture primitive definition, he defines a picture description language (PDL). The following syntax will generate any sentence S in PDL (expressed $S \in PDL$):

$$S \rightarrow P \mid (S \phi S) \mid (\sim S) \mid SL \mid (/SL) \ ,$$

$$SL \rightarrow S^{\ell} \mid (SL \phi SL) \mid (\sim SL) \mid (/SL) \ ,$$

$$\phi \rightarrow + \mid X \mid - \mid * \ ,$$

$$P \rightarrow \{\text{any primitive class name}\},$$

$$\ell \rightarrow \{\text{any label designator}\} \ .$$

For any $S \in PDL$, he defines $P(S)$ as the set of all pictures with description S. At this level a picture α is described by the pair $T(\alpha) = (T_S(\alpha), T_v(\alpha))$, where $T_S(\alpha) \in PDL$ and $T_v(\alpha)$ is a list of descriptions $D(\beta)$ of each primitive of the picture. Not only primitives, but all pictures have a tail and a head determined by their descriptions; concatenations among pictures can only occur at their tail and head positions. Consider the picture α consisting of two subpictures α_1 and α_2 such that $\alpha_1 \in P(S_1)$, $\alpha_2 \in P(S_2)$ and

$T_S(\alpha) = (S_1 \phi S_2)$, $S_1, S_2 \in$ PDL. Then the tail and head of α according to $T_S(\alpha)$ are defined:

$$\text{tail}(\alpha) = \text{tail}(\alpha_1), \quad \text{head}(\alpha) = \text{head}(\alpha_2).$$

In the same manner as primitives, more complex pictures can thus also be represented by a directed edge of a graph. This gives us the necessary hierarchy for picture processing.

The meaning of the binary concatenation operators $\{+,-,X,*\}$ is presented below by defining $P(S_1 \phi S_2)$; it is assumed that $S_1, S_2 \in$ PDL, $\alpha_1 \in P(S_1)$ and $\alpha_2 \in P(S_2)$. The notation cat means "is concatenated onto":

$$P((S_1+S_2)) = \{\alpha_1,\alpha_2 \,|\, \text{head}(\alpha_1)\text{cat tail}(\alpha_2)\},$$

$$P((S_1-S_2)) = \{\alpha_1,\alpha_2 \,|\, \text{head}(\alpha_1)\text{cat head}(\alpha_2)\},$$

$$P((S_1 X S_2)) = \{\alpha_1,\alpha_2 \,|\, \text{tail}(\alpha_1)\text{cat tail}(\alpha_2)\},$$

$$P((S_1 * S_2)) = \{\alpha_1,\alpha_2 \,|\, (\text{tail}(\alpha_1)\text{cat tail}(\alpha_2)) \text{ and } (\text{head}(\alpha_1)\text{cat head}(\alpha_2))\}.$$

The graphs of the resulting pictures are illustrated in Fig. 2.5; t and h indicate the tail and head of each expression. Note that P(S) could be empty. This is the case when the concatenations described by S are not possible according to the definitions of the primitives. For example, if ℓ_1 is a line segment primitive and $\text{head}(\ell_1) = \{(x,y) \,|\, x=C_1\}$ and ℓ_2 another line segment primitive with $\text{tail}(\ell_2) = \{x,y \,|\, x=C_2\}$, where $C_1 \neq C_2$, then $P(\ell_1+\ell_2)$ is empty. However, the graph is constructed by treating ℓ_1 and ℓ_2 abstractly.

The unary operators $\sim$ and / do not describe concatenations but allow the tail and head to be moved. $\sim$ is a tail/head reverser such that $\text{tail}((\sim S)) = \text{head}(S)$ and $\text{head}((\sim S)) = \text{tail}(S)$. The blanking or superposition

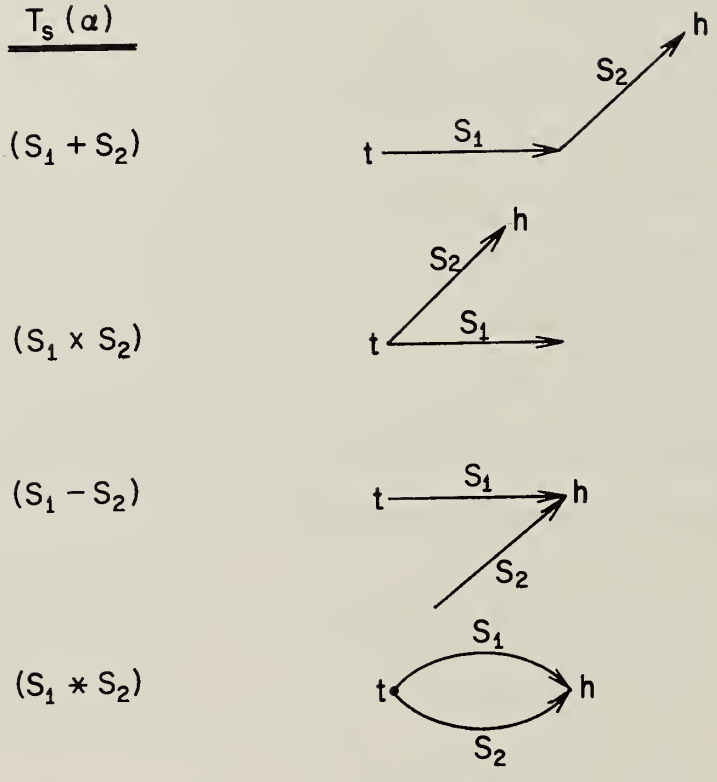


Fig. 2.5. Concatenation operators.

operator, /, works in conjunction with label designators to allow multiple appearances of the same primitive in a description, effectively relocating the tail or head on an expression. The label serves to identify the primitive or structure while the / operator indicates retracing over its associated operand.

The class of pictures of interest is described by a grammar g, and the description $D(\alpha)$ of any picture $\alpha \in P_g$ is:

$$D(\alpha) = ((T_S(\alpha), T_v(\alpha), (H_S(\alpha), H_v(\alpha)) \ ,$$

where

$T_S(\alpha) \in \mathcal{L}(g)$,

$T_v(\alpha)$ is a list of descriptions of all primitives of α,

$H_S(\alpha)$ is the parse of $T_S(\alpha)$ according to g, and

$H_v(\alpha)$ is a list of the descriptions of all non terminals in $H_S(\alpha)$.

Using PDL, the line drawing in Fig. 2.3(a) will be described as the following string language in Fig. 2.6.

We are able to describe the same line drawing of Fig. 2.3(a) in a different sentence than the one in Fig. 2.6: namely,

$$T'_S(\alpha_1) = ((((\sim d_3)+((d_1+(\sim d_4^1))*(d_2+(\sim(/d_4^1)))))*d_4^2)*(\sim(d_5*d_5))) \ ,$$

which is far more complicated than the sentences in grammar defined for the class of this line drawing, and can not be recognized as a cup easily. Again, to solve this ambiguity problem we have to depart from string languages and resort to graphical languages. Graph 2.3(c) is the graph representation of the cup, where the nodes are arc-primitives. In Fig. 2.4(b) we have shown how the structure of this line drawing can be described by a graph. Here, again, a graphical language seems to be the proper way of representing the information structures in the picture.

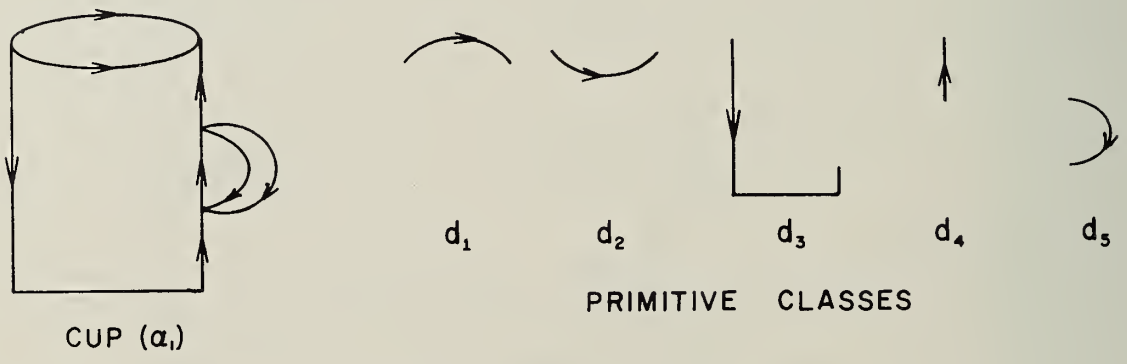
g: $P \rightarrow ELLIPSE \mid HANDLE \mid CUP$

$ELLIPSE \rightarrow (d_1 * d_2)$

$HANDLE \rightarrow (d_5 * d_5)$

$CUP \rightarrow ((\sim d_3 + ELLIPSE)*((d_4 *(\sim HANDLE)) + d_4))$

$L(g) = \{(d_1*d_2),(d_5*d_5),((\sim d_3+(d_1*d_2))*((d_4*(\sim(d_5*d_5)))+d_4))\}$



$T_S(\alpha_1) = ((\sim d_3+(d_1*d_2))*((d_4*(\sim(d_5*d_5)))+d_4))$

$H_S(\alpha_1)$:

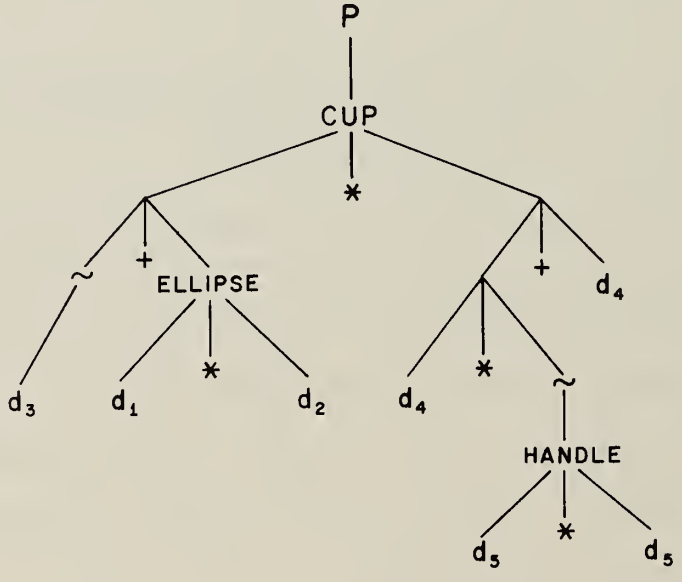


Fig. 2.6. Hierarchic description of a picture.

As to the choice of arcs as picture primitives, although the hierarchical representation is now possible, the addition and deletion of objects to the scene is awkward because arcs are not semantically as rich as regions. Of course we can use this approach positively as a preprocessing stage to our analysis in finding regions.

2.1.3 Region primitives

The last of picture primitives, and our choice is the regions, which form the "atoms" of our picture description language. These picture primitives correspond to the primitive nodes in our graph structure.

Each node of a graph represents either a closed or an open region of a picture, or can be interpreted to have a graphical structure itself. This enables us to have graphs as elements of a graph and a hierarchical scheme of presentation and interpretation of scenes. Regions are preferred to arcs as primitives, also because of their ability to have much richer semantical content, which makes the addition and deletion of objects much more feasible. As we have mentioned before, at all levels of picture analysis it is of prime importance to have all pertinent lower level information readily available to answer questions and resolve ambiguities through interrogation of this information. We have achieved this ability through association of general data structures with our graph elements. Node attributes carry the semantic information about our picture primitives, and further in a graph transformational system they can carry the semantical information which is acquired through the analysis process. The followings are a set of attributes which we have found useful in picture processing applications.

a) Shape information

Shapes represent a substantial part of semantic information about pictures. In [3] Maruyama has discussed the way, which smoothed contours of

objects can be obtained from a given black and white picture. He has introduced three basic shape representations: polygonal, pattern sequence, and skeleton representations. In our global level analysis, shape features are the most compact and also generally sufficient level of abstraction of shapes. These features can be easily extracted from Maruyama's shape representations. In Table 2.1 we have shown a set of useful shape features from his thesis.

b) Texture information

Visual texture is known to play an important role in the field of pattern recognition. In [4] Murthy has represented textures as seasonal time series. Since all we are interested in is to know the class of textures from a categorical point of view, Michalski's [2] VL_1 system (variable valued logic) is an excellent tool for representing the class of textures as a VL_L formula, and subsequently by imposition of windows over the regions of the picture and by testing this VL_1 formula, we can find the texture of the region.

c) Color attribute

It is a known fact that the colored pictures convey much richer information about the environment than simple black and white pictures. This color information can be associated with our picture elements through simple numbering of colors. However, in more sophisticated systems color attribute can be a set of color features, which can be easily extracted from pictures.

d) Size attribute

Once the scale of the picture is known, this attribute will convey important semantical information about the relative size of objects in the scene, which is essential in analyzing the 3-dimensional objects with variable orientations. Of course this attribute can be included as a feature of the

Table 2.1

Selected Shape Features

No.	Name	Description
F_1	PERIMETER	Perimeter p (unnormalized)
F_2	AREA	Area A (unnormalized)
F_3	PERIM/AREA	$(1-2\sqrt{\pi A}/p)$
F_4	M2	Degree of variance
F_5	M3	Degree of skewness
F_6	M4	Degree of elongation
F_7	MEAN-R	Mean of unnormalized elementary patterns
F_8	MAXAMP	Difference between the maximum and the minimum in elementary patterns*
F_9	DEV-R	Deviation of elementary patterns
F_{10}	ALT#	Number of alterations in a pattern sequence wrt the mean
F_{11}	MAXIMAL#	Number of local maximal points in a pattern sequence
F_{12}	DEV-E	Deviation of edge lengths
F_{13}	CONVEXITY	Degree of convexity
F_{14}	SYMMETRY	Degree of symmetry
F_{15}	DEG-AS	Degree of angular symmetry
F_{16}	DEG-F	Degree of feasibility
F_{17}	DEV-ANG	Deviation of angles
F_{18}	ANG#	Number of angles less than $\frac{\pi}{4}$ and greater than $\frac{7\pi}{4}$
F_{19}	VERTEX#	Number of vertices in a polygonal representation

shape attribute, but because of its relevant importance we have brought this up separately.

2.2 Relations as Graph Branches

In our representation a branch, directed or not directed, denotes a relation between two end nodes. Fig. 2.3(d) represents the relational graph description of Fig. 2.3(a), and by selecting a few familiar relations, the graph description of this line drawing would be as simple as Fig. 2.4(c).

The ability to associate any type of data structure with branches in a relational graph has made our method of graph representation quite general. It encompasses all the techniques discussed earlier. For example, the physical description of the border between two regions, which defines the relation between these two regions can be associated with the branch representing this relation.

Among other useful branch attributes we briefly discuss the weight attribute. Weight attribute has its values between [0,1], which further can divide each type of relation to infinite number of different relations. A 0 weight simply dictates that the branch is absent, and a weight 1 tells us that the branch must exist. In other words the weight attribute value can be interpreted as the probability of the branch existence. In a graph descriptional representation of pictures, labeled branches are mainly used to transfer the positional relations of the regions to the analyzer. Although it may seem that a single label information is hardly sufficient to represent complex relational informations, as we mentioned above, we have the ability to associate with each branch any additional information that we might need in different applications.

Through this research we have discovered that the existence of certain relations with their known properties is of a great help in analyzing pictures.

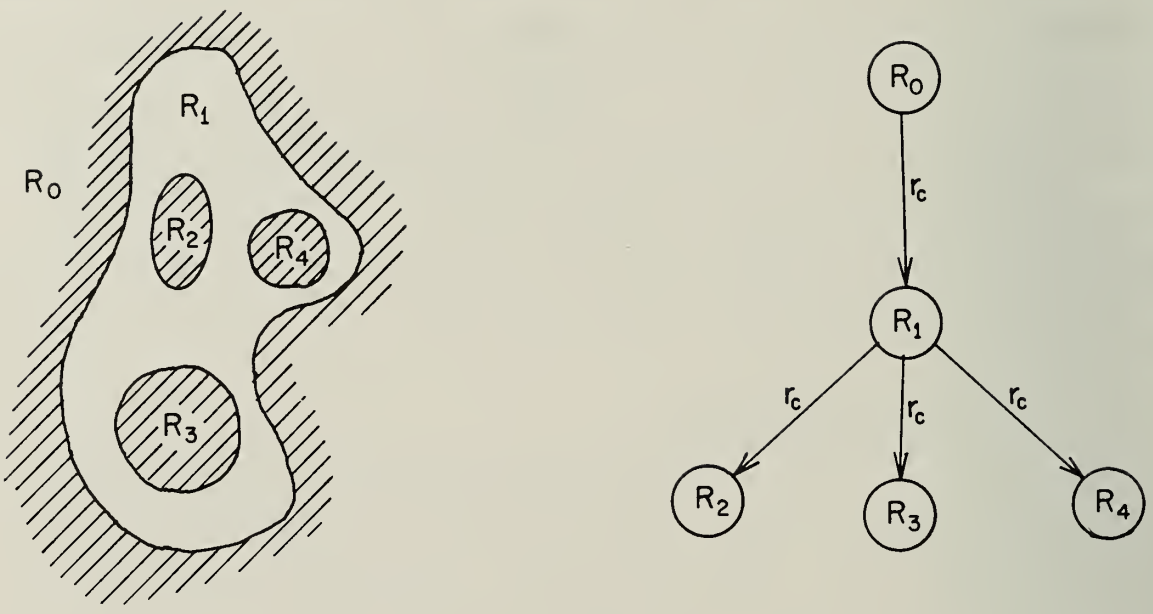
Since the actual extraction of these relations from array representation of the scene is the task of the preprocessor and is highly dependent on the way each region is presented, we do not get into details of how this should be done. Maruyama [3], in connection with shape representations, has discussed the extraction of several useful relations. To make the material self-contained we include here some of his techniques for relation extraction.

2.2.1 Containment relation

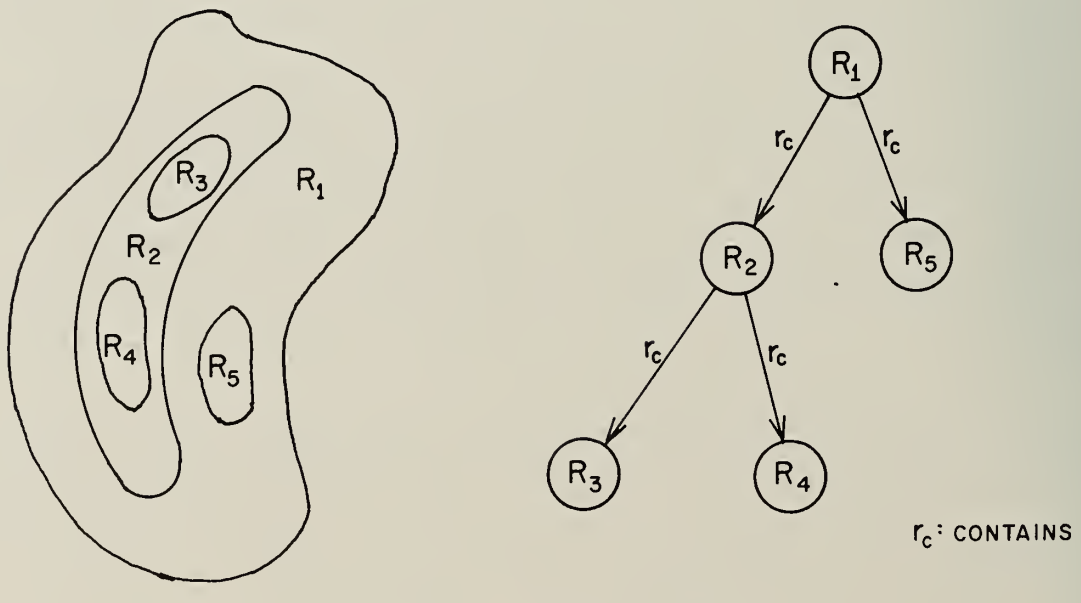
The topology of a plain black and white picture or regional picture can be simply described by a tree, a containment tree. For example, the topological relation--"contains," "inside"--of pictures illustrated in Fig. 2.7 can be described simply by the containment trees as illustrated, where each node refers to a connected component of the figure or closed region. Thus such a tree is a subgraph of the graph which describes the structure of the picture.

For array pictures, Buneman (1970) presented an algorithm which determines this containment tree from information about the picture which is picked up during a single scan of array. His algorithm strongly resembles a generative grammar and a procedure for choosing the rule of the grammar to operate.

His algorithm of generating containment trees assumes that the given picture is described as a set of polygonal curves. He also assumes that for each such closed curve a point inside the curve, the center of gravity or the AS-point (angulary simple), is available. The procedures for deriving trees is quite similar to the determination of intersections between a straight line and with a set of curves. He emanates a ray from each given point which is inside the corresponding region and determines a sequence of



(a) black and white array picture



(b) region bounded picture

Fig. 2.7. Containment trees.

intersections with the ray and the regional boundaries. Without loss of generality, rays are emanated along the X-coordinate. A simple analysis of the generated intersection sequence shows the possible containment relations.

Let us consider the picture of Fig. 2.8, which consists of five closed regions. The regions are denoted by R_1 through R_5, and their interior points are X_1 through X_5, respectively. For example, region R_5 is contained in R_2, and, in turn, R_2 is contained in R_1. This can be determined by emanating a ray at X_5 and examining the sequence of boundary intersections. In this case, the sequence is R_5, R_2, R_1, and is an ordered sequence. It is interpreted as for each pair of adjacent elements, the left element contained in the right element. Thus, $R_5 \subset R_2$ and $R_2 \subset R_1$.

Let us consider region R_3. Its sequence is R_3, R_5, R_5, R_2, R_1. This means the emanated line at X_3 doubly crosses region R_5, which clearly implies that X_3 is not contained in R_5; consequently R_3 is not contained in R_5. Thus any pair of identical elements should be eliminated from the sequence. After this shrinking process, the sequence becomes R_3, R_2, R_1 and is interpreted as $R_3 \subset R_2 \subset R_1$. The region R_1 has its corresponding sequence as R_4, R_2, R_1. However, since we are considering and determining which regions contain R_1, the final sequence should be R_1.

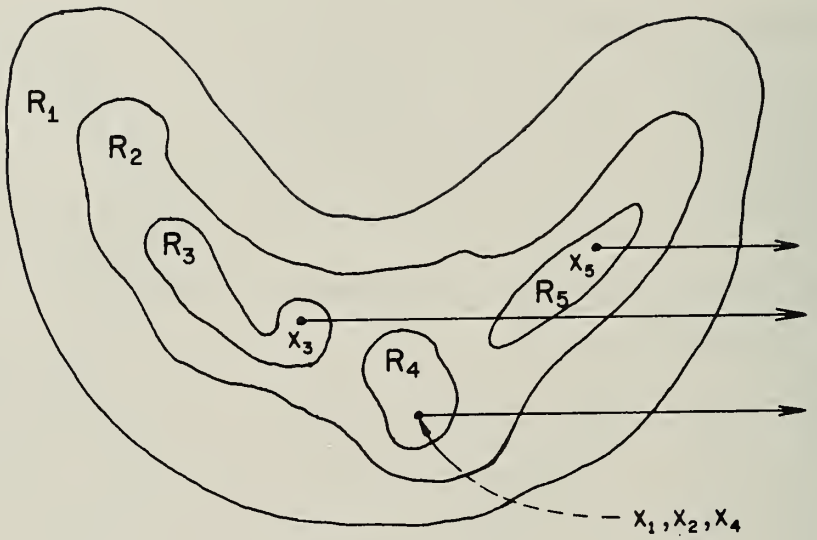
From the above examples, the following simple grammar is driven. Let R be a sequence,

$$R = R_{i1}, R_{i2}, \cdots, R_{ij}, \cdots, R_{in}$$

generated at a point x_k of the region R_k. Let us assume that symbols α, β, γ denote any sequences including null sequence. Then the grammar consists of the following two rules:

RL1. $\alpha R_{ij} \beta R_{ij} \gamma \rightarrow \alpha\beta\gamma$

RL2. Let x_k be the point of R_{ij}, $\alpha R_{ij} \beta \rightarrow R_{ij} \beta$.



(a)

X_1: R_4, R_2, R_1

X_2: R_4, R_2, R_1

X_3: R_3, R_5, R_5, R_2, R_1

X_4: R_4, R_2, R_1

X_5: R_5, R_2, R_1

(b)

X_1: R_1

X_2: R_2, R_1

X_3: R_3, R_2, R_1

X_4: R_4, R_2, R_1

X_5: R_5, R_2, R_1

(c)

$R_2 \subset R_1$

$R_3 \subset R_2$

$R_4 \subset R_2$

$R_5 \subset R_2$

(d)

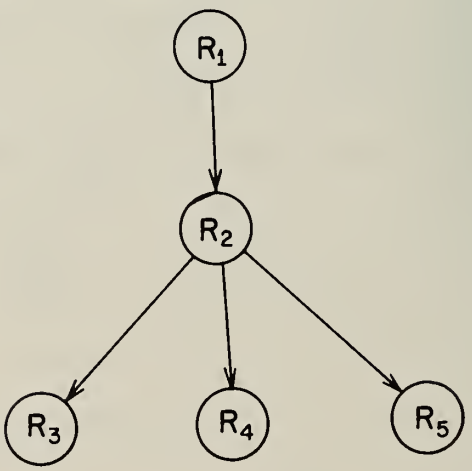


Fig. 2.8. An algorithm to generate containment trees.

For any generated subsequence (or complete sequence), RL1 is applied as many times as possible, and then, finally, RL2 is applied.

In Fig. 2.8(b), generated sequences at X_1 through X_5 are listed, (c) shows the results obtained after the application of rules RL1, RL2, and final containment relations as well as containment tree of picture (a) are illustrated in (d). In (d) only direct containment relations are exhibited, and transitive containment relations such as $R_3 \subset R_1$ are eliminated.

2.2.2 Neighborhood relations

Relations, such as "next"; "above" or "below", "above and touching" or "below and touching"; "left of" or "right of", "left of and touching" or "right of and touching"; "behind" or "in front of", "behind and touching" or "in front of and touching", are considered to be heighborhood relations. These relations can be defined only between brother regions, where regions which have an identical direct father in the containment tree are said to be brothers. Thus, after construction of a containment tree for a given picture, we know which regions of the picture are brothers.

The determination of relations between a pair of neighboring regions is as simple as the construction of a containment tree. Let us assume that the regions $R_1, R_2, \cdots, R_j$ and R_n are brothers, and let us assume that we are determining the relations between a region R_j and the rest of regions $R - \{R_j\}$, where

$$R = \{R_1, R_2, \cdots, R_j, \cdots, R_n\} .$$

We also assume that $x_j \in R_j$ for $j=1,\cdots,n$. To determine relations, a generalized primal pattern sequence is generated about x_j within the space of $R-\{R_j\}$, here each elementary pattern has as an attribute, the sequence of regions which have been pierced. Thus, for this type of pattern sequences about x_j,

we may ignore measuring the distance between x_j and the intersection of elementary pattern with a region; however, we detect the region name and whether or not this intersection point is on the boundary of both regions.

In Fig. 2.9, we have illustrated the process of determining these relations. Fig. 2.9(a) shows a picture of eight regions and its containment tree. From the tree, regions R_5, R_6, R_7 are brothers, and their father is R_4. Only regions that are brothers are illustrated in (b), and their possible relation graph is exemplified. In (c) a picture of three regions is shown, and the process of determining relations between R_2 and $R - \{R_2\}$ is illustrated as the generation of primal pattern sequence. From the sequence we can determine all possible relations. The sequence generated at X_2 is

$$S(X_2) = S_0, S_1, \cdots, S_{12}$$
$$= 1,1,1,0,0,0,0,0,0,0,0,1,1,$$

where 0 denotes empty space, 1 denotes R_1. We also have the information whether these intersections are on the boundary of both regions or not. From the sequence $S(X_2)$ and this information, it becomes clear that R_2 is not connected to R_1 or R_3, and further from the indices of elementary patterns we derive: R_2 is at the left of R_1.

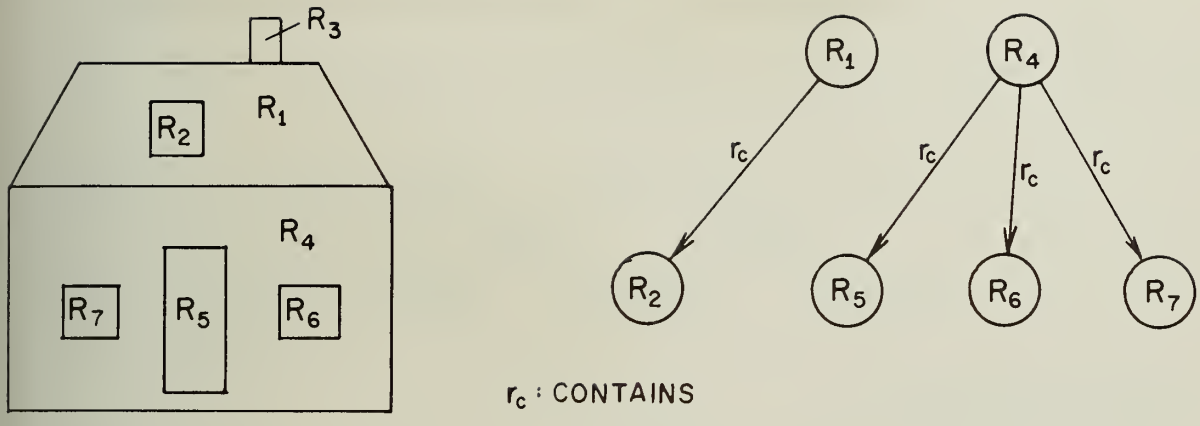
In general, in case of two dimensional pictures, $S(X_j)$ can be divided into four regions.

$$S(X_j) = S_0, S_1, \cdots, S_{N/2}, \cdots, S_{N-1}$$

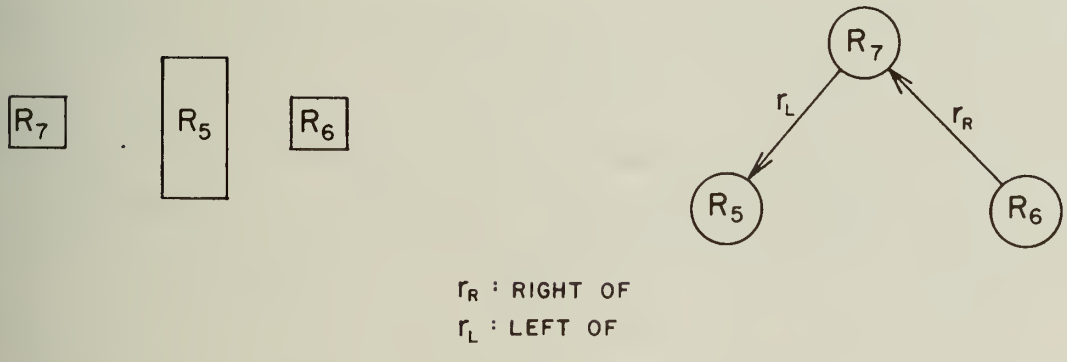
below (over $S_1, \cdots, S_{N/2}$); above (over $S_{N/2}, \cdots, S_{N-1}$)

left of (over S_0, S_1); right of (over $S_1, \cdots, S_{N/2}$); left of (over $S_{N/2}, \cdots, S_{N-1}$)

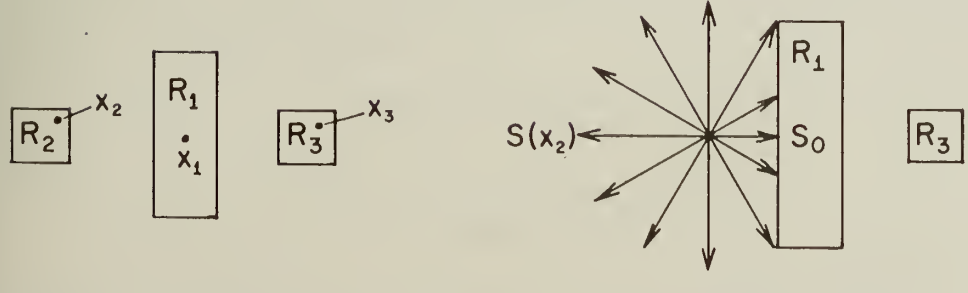
From these illustrations, it should be clear that much more sophisticated relations could be discovered by examining these kinds of pattern sequences.



(a) A picture and its containment tree



(b) Brotherhood regions and their relation graph



(c) $S(x_2) = S_0, S_1, \cdots, S_{12}$ $\quad S_i \in \{R_1, R_3, \phi\}$

Fig. 2.9. An algorithm to generate neighborhood relations.

In the case of three dimensional pictures we can have another sequence of rays, $S'(X_j)$, which their plan is perpendicular to the plan of $S(X_j)$.

$$S'(X_j) = S_0', \underbrace{S_1', \cdots, S'_{N/2}}_{\text{below}}, \underbrace{\cdots, S'_{N-1}}_{\text{above}}$$

back of — front of — back of

2.3 Graph Definition

Before getting into formal definition of our graph-structure, it would be in order to mention WEBS as immediate predecessor of our system. In [16] J. L. Pfaltz defines a web as follows:

A directed graph G = (S,R) is a set S of points (or nodes) together with a relation R on S. Recall that a relation on a set S is a subset of SxS, that is a set of ordered pairs $\{(a,b) | a,b \in S\}$. R is usually called the set of arcs (or edges) of G. If there exists a function λ: $S \rightarrow V_S$, where V_S is any set of symbols ("vocabulary") then G is said to be a point-labeled graph over V_S. If there exists a function μ: $R \rightarrow V_R$ for some set of symbols V_R, then G is said to be an edge-labeled graph over V_R. Point or edge labeled graphs are called webs.

Pfaltz and Rosenfeld [21] have shown that the concept of a phrase-structure grammar can be extended in a natural way from strings to webs. In our graphs we have both nodes and branches labeled.

Let,

$N = \{n_1, n_2, \cdots, n_m\}$ set of nodes,

$B = \{b_1, b_2, \cdots, b_p\}$ set of branches,

$A = \{a_1, a_2, \cdots, a_v\}$ set of attributes,

$U = \{u_1, u_2, \cdots, u_r\}$ set of labels for branches.

Then a graph g_i is defined as:

$$g_i = \{N_i, B_i | A_i, U_i\}$$

$$G = (g_1, g_2, \cdots, g_q) \qquad \text{set of graphs}$$

where,

$$N_i \subset N ,$$

$$B_i \subset B ,$$

$$A_i \subset A ,$$

$$U_i \subset U ,$$

and since each branch, $b_i = \{n_j, u_k, n_\ell\}$ or

$$B \subset N \times U \times N ,$$

then N_i must contain all the nodes of the branches in B_i. m is the maximum number of nodes allowed by implementation.

Definition: g_i is a subgraph of g_j if and only if $g_i \subseteq g_j$.

A is a set of attributes for nodes, branches, and graphs;

$$A = Na \cup Ba \cup Ga,$$

each a_i is a function,

$$a_i : \{N | B | G\} \rightarrow v_i ,$$

where, v_i is a set of values.

Equivalently, we may consider the branches as a set of relations $\{Ru_1, Ru_2, Ru_3, \cdots, Ru_r\}$ where,

$$Ru \subset N \times N ,$$

and a branch $b_i = \{n_j, u_k, n_\ell\}$ exists if and only if

$$(n_j, n_\ell) \quad Ru_k .$$

Let $G = \{g_1, g_2, \cdots, g_q\}$ represent a set of graphs, where g_i is defined as above. The following functions are defined to operate on the elements of the graph or the graph themselves.

TAIL: $B \rightarrow N$, $TAIL(n_j,u_k,n_\ell) = n_j$

HEAD: $B \rightarrow N$, $HEAD(n_j,u_k,n_\ell) = n_\ell$

LABEL: $B \rightarrow U$, $LABEL(n_j,u_k,n_\ell) = u_k$

NODES: $G \rightarrow N$, $NOF(g_i) = \{n_j | n_j \in g_i\}$

BRANCHES: $G \rightarrow B$, $BOF(g_i) = \{b_j | b_j \in g_i\}$

ADJBRS: $N \rightarrow 2^B$, $ADJBR(n_i) = \{b_j | b_j = (n_i,u_k,n_\ell) \text{ or } (n_\ell,u_k,n_i)\}$

INCBRS: $N \rightarrow 2^B$, $INCBR(n_i) = \{b_j | b_j = (n_\ell,u_k,n_i)\}$

OUTBRS: $N \rightarrow 2^B$, $OUTBR(n_i) = \{b_j | b_j = (n_i,u_k,n_\ell)\}$

NAME: $\{N|B|G\} \rightarrow L$ where L is a set of labels and $U \subseteq L$

These are a few of the functions which operate on the graph elements. See Appendix A about other functions.

2.4 Linguistic Description

As we have explained so far, although string languages can be useful in preprocessing stages, they are inadequate for global analysis of pictures. Our structure operation language can be used to declare graphical representation of the scenes statically, or by the help of preprocessing procedure to create graphs dynamically. In Fig. 2.10(a) we have an example of a scene in which the graphical representation is given pictorially in (b) and linguistically in (c) as a SOL declaration. Here, the branch labels have the following interpretations:

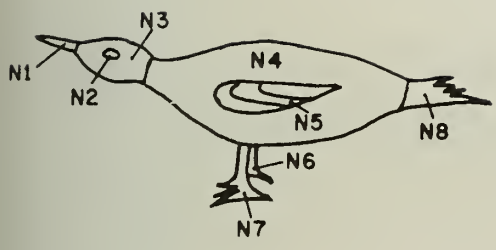
DLOF: Directly left of

DROF: Directly right of

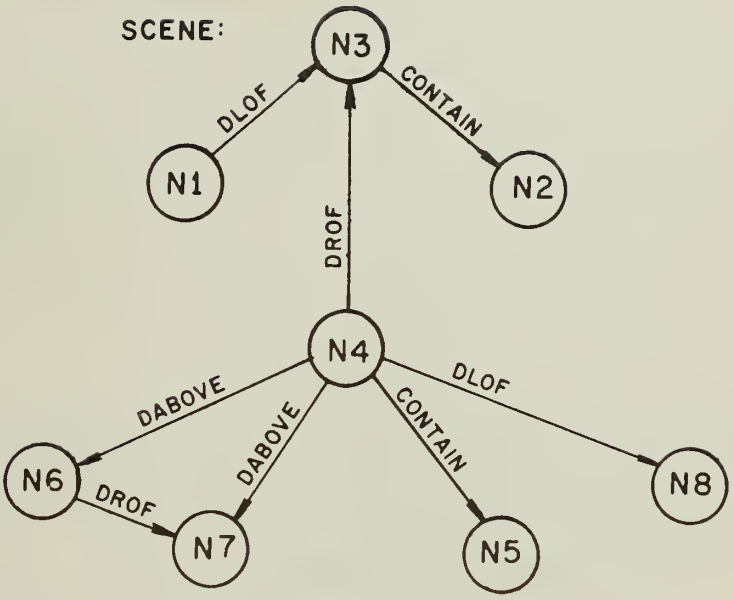
CONTAIN: Contains

DABOVE: Directly above.

For example, $N_1 \xrightarrow{DLOF} N_3$; means, region N_1 is located directly on the left of region N_3.



(a) scene



(b) relational graph

```
DCL SCENE GRAPH,
    DLOF BR  (N1,N3),
    DROF BR  (N4,N3),
    CONTAIN BR (N3,N2),
    DABOVE BR (N4,N6),
    DABOVE BR (N4,N7),
    DROF BR (N6,N7),
    CONTAIN BR (N4,N5),
    DLOF BR (N4,N8);
```

(c) SOL declaration of the scene graph

*Associate statements are used to assign the values of node attributes.

Fig. 2.10. Linguistic description of a scene.

As we mentioned before, there can be any kind of general data structures associated with any element of our graph, and these are attributes which convey semantical information about the picture.

For example, if attribute a_i conveys the shape information about primitive regions of the picture and it has the following PL1 structure,

```
DCL 1  AI,
       2  F(10) BIN FIXED,
       2  NAME CHAR(8),
```

then we can associate this structure with all the nodes of the graph in 2.10(b) with the following SOL statement:

```
NDASOC (NOF(SCENE))  1  AI,
                     2  F(10) BIN FIXED,
                     2  NAME CHAR(8);
```

now we have ten shape attribute features and one eight character string associated with every node in graph 2.10(b). Attributes can be used both as functions and as pseudo variables. For example, if the following structure V has the values to be attributed to node N_3 of graph "scene",

```
1 V,
2 ARRAY (10) BIN FIXED,
2 STRING CHAR(8);
```

then it can be assigned with the following statement:

```
AI(SCENE.N3) = V,
```

and the values of the attributes can be retrieved in the same manner

```
V = AI(SCENE.N3).
```

For detailed description of the language see Appendix A. Pointers are extensively used when the name references are ambiguous.

3. GRAPH TRANSFORMATIONS

In chapter 2 we showed how scenes can be represented in a natural way through graphical description. Inferring from a conceptual model of the universe, graph transformations are applied to the scene graphs to transfer them to more readily interpretable structures. Transformations also can act on the model objects of the universe to produce variations of these models.

Graph transformations are formally defined in the same manner as in [17] by Schwebel: A graph transformation, T, between two graphs g_1 and g_2 is a relation $T \subseteq g_1' x g_2'$ where $g' = g \cup \lambda$. Here we assume an element, λ, called the empty element, which is associated with each graph, g. An element of g_1, mapped into λ_2 only, is said to be deleted by the transformation and an element of g_2 which is the image of λ_1 only is said to be created by the transformation.

The inverse of a graph-structure transformation T is denoted T^{-1} and is the inverse of the relation. That is,

$$T^{-1} = \{(y,x) \mid (x,y) \in T\} .$$

Note: Delete is the inverse of create.

In general in our picture processing application we are interested in the contracting transformations--many to one--i.e., $|g_1| > |g_2|$.

3.1 Domain and Range of Transformations

Transformations act on a graph (nodes and branches), and produce a graph (nodes and branches). Because we wish to allow contextual elements to be included in the domain or range of the transformation, the relation need not be everywhere-defined nor onto. In fact, those parts of the graph which are not affected by transformation remain intact as parts of the newly formed graph.

Fig. 3.1(a) is an example of a scene; (b) is its graphical description; (c) defines a transformation T_1 which maps nodes and branches into a branch, and nodes and branches into a node; (d) defines another transformation, T_2, which maps branches into a branch, and nodes and branches into a node.

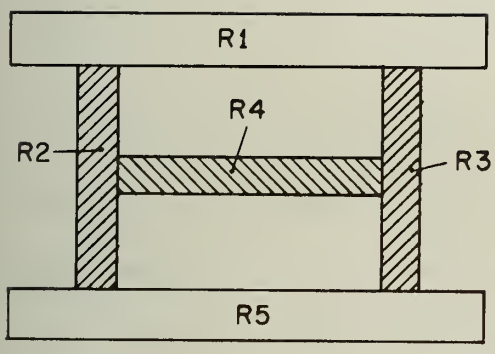
The empty element, λ, is introduced to specify creations and deletions. We could have defined unmapped elements to be deleted, but that would have eliminated the possibility of having elements in the graph which are present only to establish context and which are not mapped or deleted.

Definition: domain of a transformation is that part of the graph which is mapped or deleted. Range of transformation is that part of the newly formed graph which the elements of the domain are mapped into.

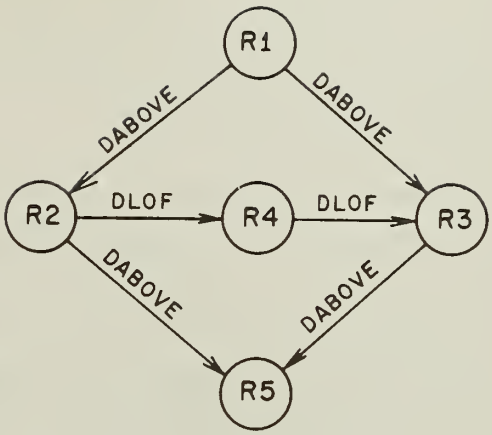
The element mapping and the presence of the empty element in a transformation also allow us to define the memory of an element in the range of a transformation. This is a useful concept to have when considering conditions for performing the inverse transformation and to simplify the representation of a series of graph transformations. The memory, denoted M(e), of an element e which is in the range of the transformation T_j is the inverse image of e under T_j, that is, $M(e) = T_j^{-1}(e)$. Normally in a series of transformations, T_j will be the last transformation which had e in its range.

Then after a transformation, $T \subseteq g_1' x g_2'$ to a graph $G(g_1 \subseteq G)$, we have available the transformed graph $G'(g_2 \subseteq G')$, which contains the range of T, and the history of the transformation, represented by the memory of the elements of g_2: M(e) for $e \in g_2$, ($e \neq \lambda_2$ if e is a node). This simplifies the representation of g_2, which will then be the domain for further transformations.

One problem which arises in a contextual transformation is the fate of relations between the nodes in the domain and other nodes not in the domain.



(a) scene



(b) graphical representation

(c) (node,branch) → branch
(node,branch) → node
transformation T_1

$T_1(R1) = R1$

$T_1(R5) = R5$

$T_1\{(R1,DABOVE,R2),(R1,DABOVE,R3),$
$(R2,DLOF,R4),(R4,DLOF,R3),$
$(R2,DABOVE,R5),(R3,DABOVE,R5),$
$R2,R4,R3\} = (R1,DABOVE,R5)$

(d) (node,branch) → node
(branch) → branch
transformation T_2

$T_2(R1) = R1$

$T_2(R5) = R5$

$T_2((R1,DABOVE,R2),(R1,DABOVE,R3))$
$= (R1,DABOVE,R')$

$T_2(R2,DABOVE,R5),(R3,DABOVE,R5))$
$= (R',DABOVE,R5)$

$T_2((R2,DLOF,R4),(R4,DLOF,R3),$
$R2,R4,R3) = R'$

Fig. 3.1. Examples of graph transformations.

Associated with each transformation, T, there must be another transformation which is a function of these branches as well as T, and maps these branches to another set of branches and the empty element λ. Let us call this an embedding transformation ET.

In Fig. 3.2, branch (a_1,A,a_5) is mapped into branch (a_1,A,n_2) and branch (a_1,B,a_2) is mapped into branch (a_1,λ_2,n_1); in other words, deleted. The graphs g_1 and g_2 are encircled and are referred to as the domain sphere and the range (or transformed) sphere, respectively, of T. The memory of n_1, $\{a_2,a_3,(a_2,C,a_3)\}$ is shown cross-hatched.

3.2 Parsing vs. Transformation

Let us impose the following restrictions on the transformations.

1. Branches and nodes are mapped into nodes, and for every branch mapped into a node, its end nodes must also map into the same node.
2. Only branches are mapped into a branch.
3. The mapping is a functional mapping, i.e. many-into-one and not one-into-many.

Theorem 1: With the above restrictions, we can represent every transformation as a series of transformations with the following characteristics:

a) range of each transformation is a single node, so that every node in the range of T defines a transformation.

b) for each of the above transformations the embedding branches are defined the same way as in T or as in its embedding function ET, whichever may be the case.

Proof: The proof is straightforward. Assume $G_i = (N_i,B_i)$ and

$$B_i = B_{i1} \cup B_{i2} \cup B_{i3} \cup B_{i4}$$

B_{i1}: set of branches cutting the domain of T

B_{i2}: set of branches mapping into a node

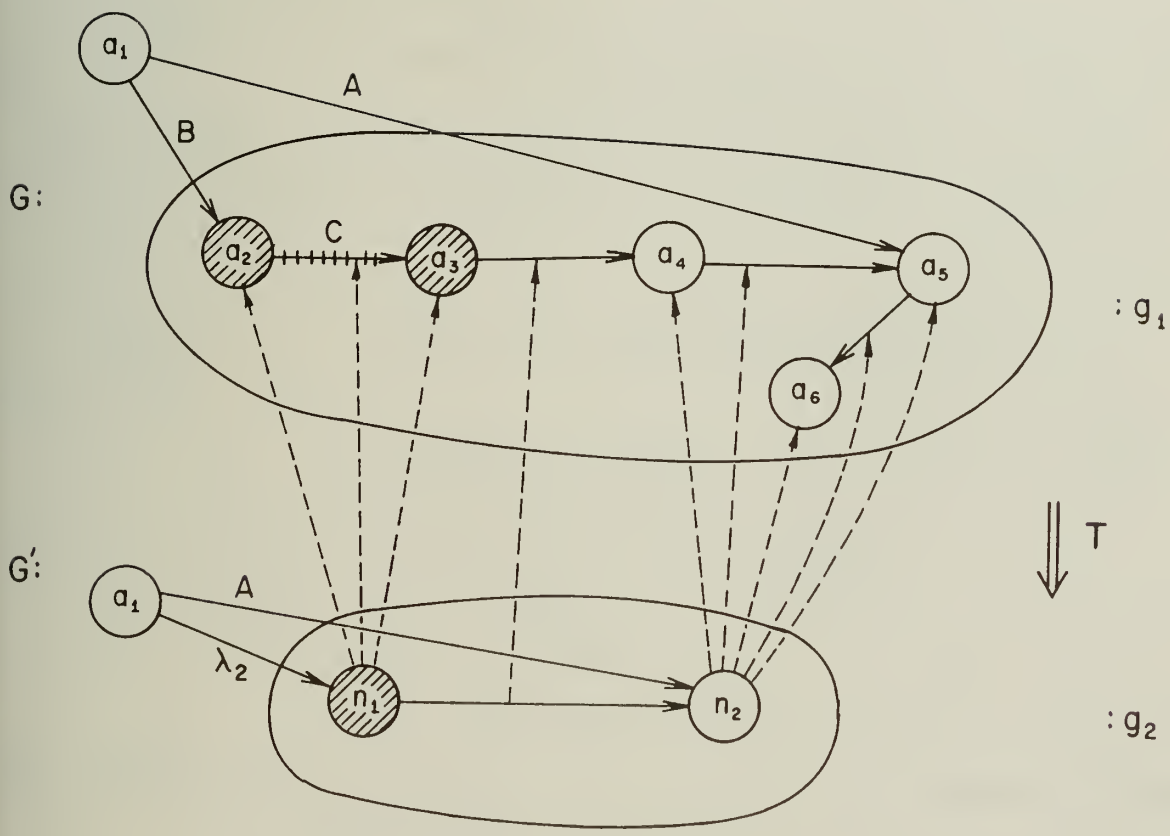


Fig. 3.2. Illustration of transformation domain and range and memory.

B_{i3}: set of branches mapping into a branch

$B_{i4} = B_i - \{B_{i1}, B_{i2}, B_{i3}\}$.

Let m be the number of nodes in the range of transformation T, then we show that,

$$T = T_1 T_2 \cdots T_m,$$

where T_i is in the class of transformations defined above.

Each T_i is a mapping function, which maps the elements from the domain of T to the node corresponding to T_i. We define the embedding function of T_i as follows:

for each branch cutting the domain of T_i either,

$b \in B_{i1}$

then $ET_i(b) = ET(b)$ ET_i: embedding function for T_i,

or $b \in B_{i3}$

in which case

$ET_i(b) = T(b)$.

In all other cases we do an identity embedding.

Identity embedding: If $b = (a_i, u, a_j)$ or $b = (a_j, u, a_i)$, where a_i belongs to the domain of T_i, and a_j is outside of its domain then $IE(b) = (n_i, u, a_j)$ or (a_j, u, n_i) correspondingly. n_i is the range (node) of T_i.

Fig. 3.3 shows the split of transformation T in Fig. 3.2 into two transformations T_1 and T_2.

With the above three simple restrictions on transformations and theorem 1, we have shown that our class of transformations which have a single node range are quite general, and this provides us with a built-in hierarchy, which facilitates the inverse transformations (T^{-1}) at single node level.

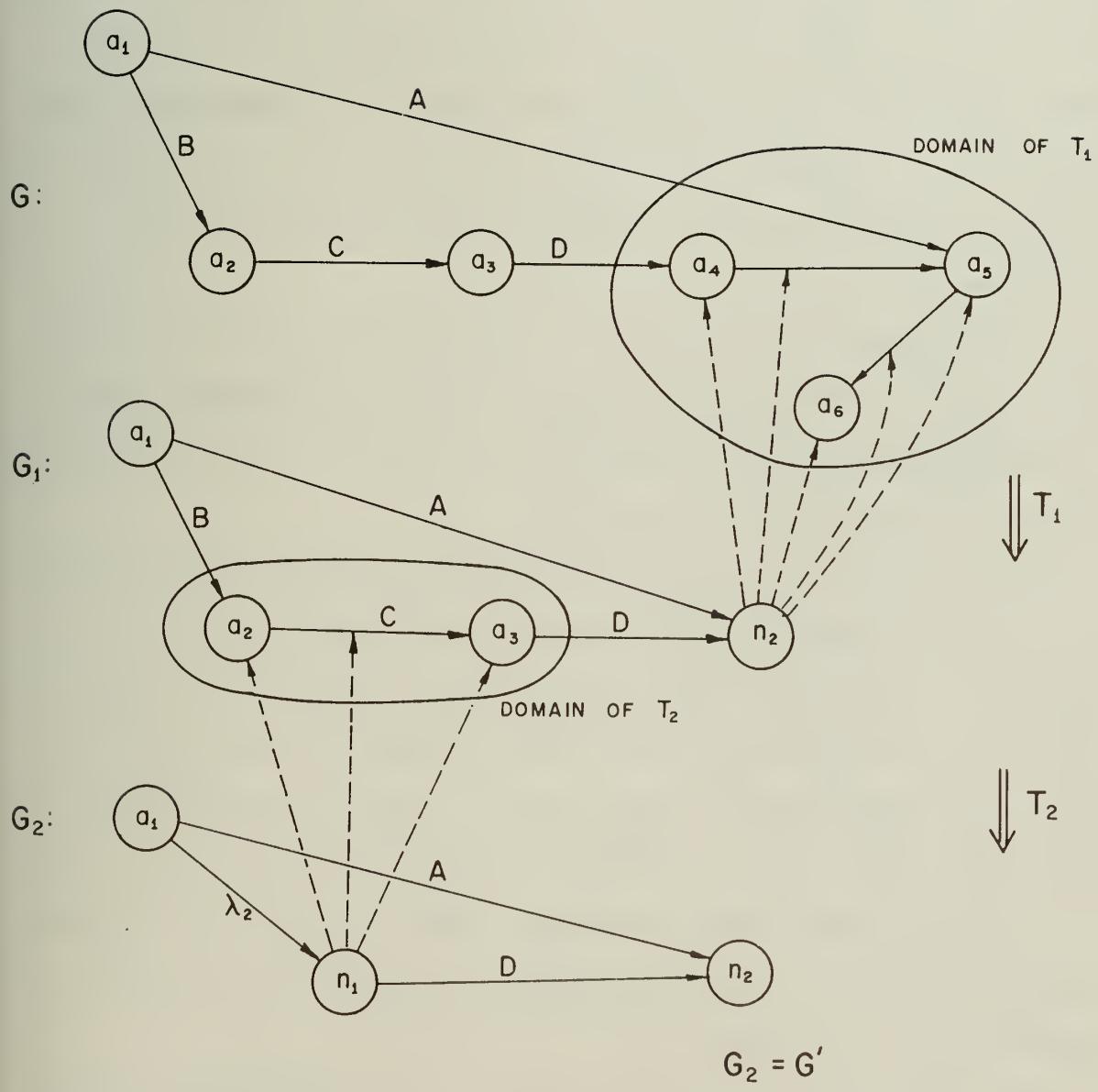


Fig. 3.3. Implementing transformation T into steps T_1 and T_2.

3.3 Validity of Embedding Relations

An application of a transformation is valid if the resulting graph is valid as a representation of a physical situation. This validity will depend upon the logical definitions of the system. For example, if composites represent unions of set elements, and branches represent binary relations between elements, then the validity of the transformation depends upon the validity of the relations created by the transformation. We can make a transformation valid by defining a proper embedding function for the transformation.

T is reversible if and only if T is valid and T^{-1} is valid, given the memory of T. A reversible transformation is said to be information lossless. A reversible transformation is one which can be validly inverted assuming the memory of the transformation is available.

One important criterion in selecting our class of transformations is the fact that the memory of a transformation can be saved entirely in the single node of the range. If our embedding function dictates that an embedding relation can not exist, we will create a temporary branch which its memory is the branch which had to be deleted. This gives a full reversibility to the class of transformations which we defined in 3.2.

In parsing graphs, a back-up procedure, which restores the environment exactly as it was before the transformation was applied, is an absolutely necessary part of the system.

The values of embedding function depends on the value of the variables in a transformation (nodes and branches in the domain and their attributes; branches cutting the domain of transformation and their attributes). Fig. 3.4(a) shows that for transformation T, binary relation "contain" is

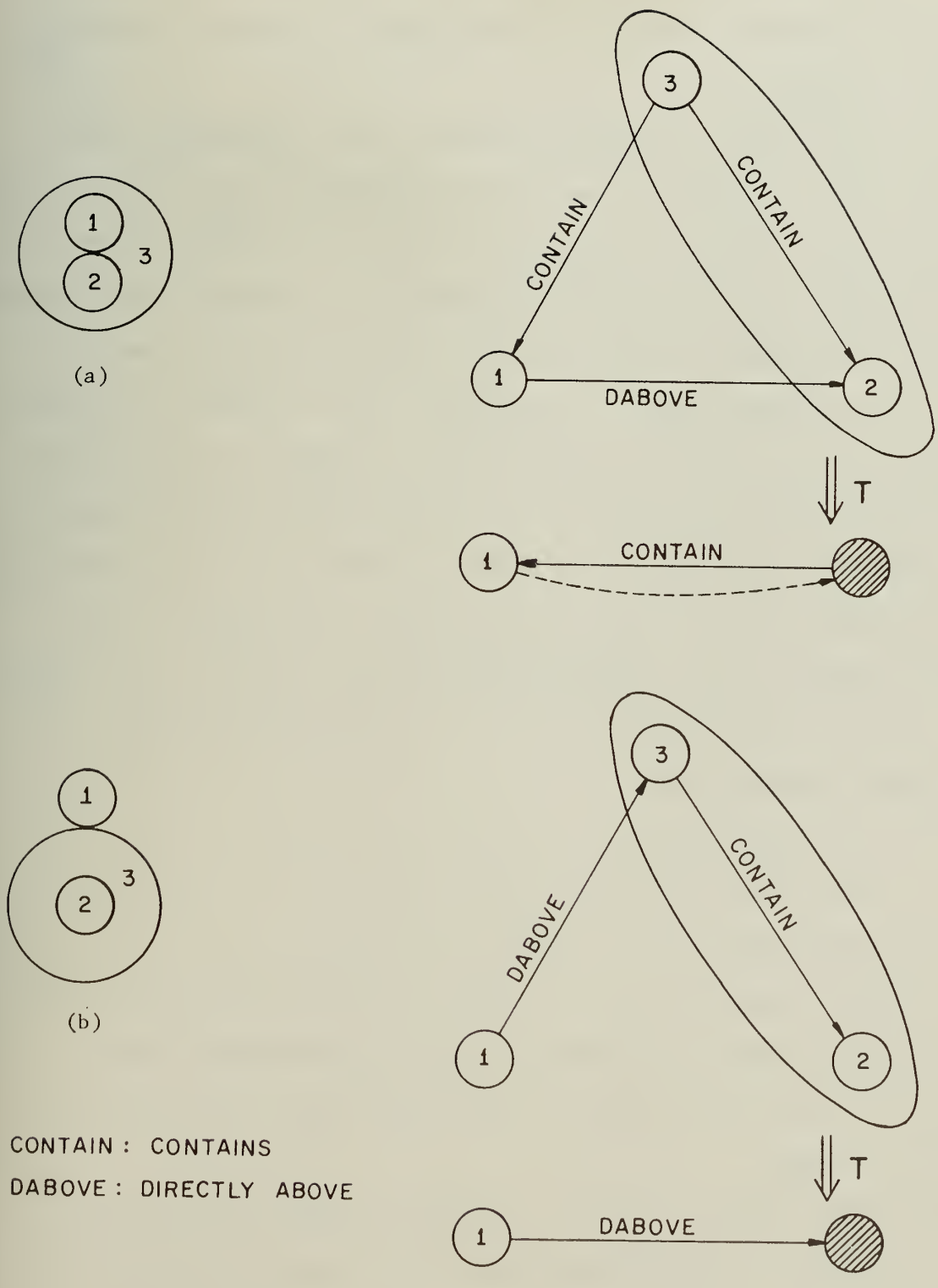


Fig. 3.4. An example of embedding branch depending on the variables of domain.

preserved, while "DABOVE" had to be deleted. In 3.4(b) the same "DABOVE" relation is preserved, because now we have branch (1,DABOVE,3) instead of branch (1,DABOVE,2) in (a).

In [17] Schwebel has considered general rules for embedding a graph g_2, which has resulted from a transformation $T \subseteq g_1' x g_2'$, into the graph of the domain g_1. He has considered rules as properties of relations for arbitrary transformations, which we know is not generally true for most common relations and has to be defined in connection with transformation and the branch itself , not only branch labels. His embedding rules determine only whether a branch (n,u,m), where n is outside of and m is within the domain sphere, implies a new branch (n,u,t) where t is the range node and inside transformed sphere. The decision as to whether a branch exists depends upon the inverse image, $T^{-1}(t)$, of the g_2 and its context. He defines the three following embedding types. Assume $n,m \in N$, $n \notin g_1$, $m \in g_1$, $u \in U$, $t \in T(m)$, $t \neq \lambda$ (in our class of transformations t is the only element of g_2).

Simple Embedding Type

OR (n,u,m) For any $m \in T^{-1}(t) \Rightarrow (n,u,t)$

AND (n,u,m) For all $m \in T^{-1}(t) \Rightarrow (n,u,t)$

NOT (n,u,m) For no $m \in T^{-1}(t) \Rightarrow (n,u,t)$.

Thus, embedding types allow us to determine connections from an element outside the domain sphere to the elements within the transformed sphere as logical functions of the elements of the inverse image of the new element and their context.

Let NODES $(T^{-1}(t)) = \{m_1, m_2, \cdots, m_k\}$ and $b = (n,u,t)$, $b_i = (n,u,m_i)$, as shown in Fig. 3.5. Let f_i be a predicate which is true if and only if branch b_i exists; similarly, f is a predicate which is true if and only if the branch exists. Then the embedding types, OR, AND, and NOT are expressed in

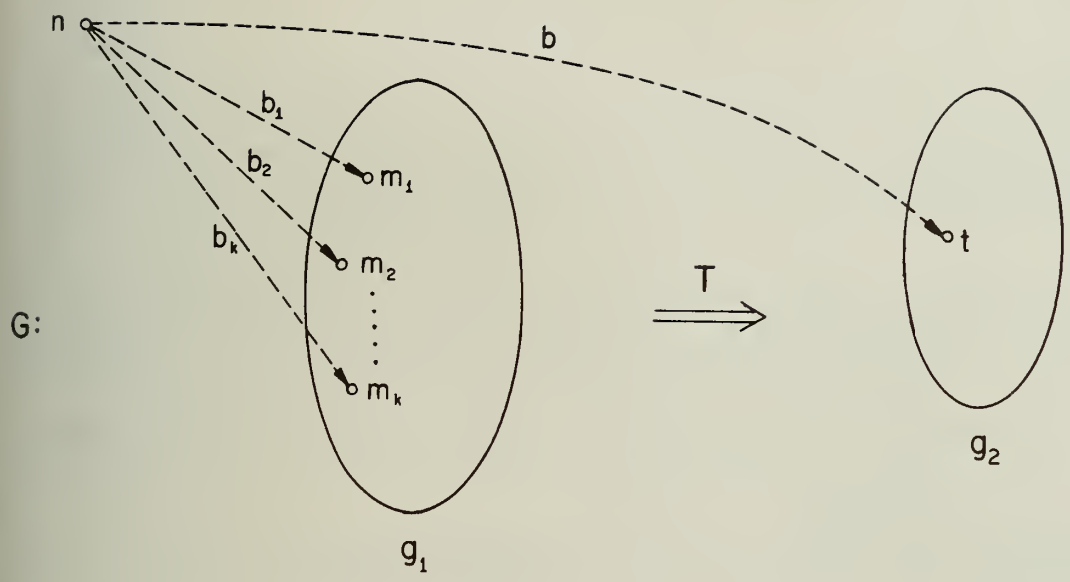


Fig. 3.5. Simple embedding types.

terms of predicates as given below.

$$\text{OR} \quad \bigvee_i f_i \Rightarrow f$$

$$\text{AND} \quad \bigwedge_i f_i \Rightarrow f$$

$$\text{NOT} \quad \neg(\bigvee_i f_i) \Rightarrow f \ .$$

But as we mentioned before, these are defined as the property of relations and very few relations have this property. The best way to investigate about this is to define embedding functions with a set of actual relations.

3.4 One Class of Relations

In Table 3.1 we have shown a set of relations which are very useful in picture processing. This set of relations are closed under inversion i.e. if relation r is in the set, so is r^{-1}.

Table 3.2 shows some of the basic properties of these relations. In [22], B. H. McCormick has made an extensive representation of properties for this type of relations.

3.4.1 Embedding function for this class of relation

As we have learned, the value of embedding function not only depends on the branches which cut into the domain of transformation, but also depends on the variables of transformation. Here, we study the case where embedding function depends on the properties of the binary relations in the domain and the ones which cut the boundary of the domain.

We use the same nomenclature as in Fig. 3.5.

NEXT: node n must be in relation "NEXT" to a node in the region which is not inside any other node (object) of the domain;

Table 3.1

A Class of Relations

Relation		Explanation
NEXT	1	tail node object and head node object are touching in more than one point $(\text{NEXT})^{-1} = \text{NEXT}$
CONTAIN	2	tail node object contains head node object $(\text{CONTAIN})^{-1} = \text{INSIDE}$
INSIDE	3	tail node object is inside the head node object $(\text{INSIDE})^{-1} = \text{CONTAIN}$
LOF	4	tail node object is located at the left of head node object and not touching $(\text{LOF})^{-1} = \text{ROF}$
ROF	5	tail node object is located at the right of head node object and not touching $(\text{ROF})^{-1} = \text{LOF}$
ABOVE	6	tail node object is located above head node object and not touching $(\text{ABOVE})^{-1} = \text{BELOW}$
BELOW	7	tail node object is below head node object and not touching $(\text{BELOW})^{-1} = \text{ABOVE}$
INB	8	tail node object is located in the back of head node object and not touching $(\text{INB})^{-1} = \text{INF}$
INF	9	tail node object is located in front of head node object and not touching $(\text{INF})^{-1} = \text{INB}$
DLOF	10	tail node object is located at the left of head node object and touching $(\text{DLOF})^{-1} = \text{DROF}$
DROF	11	tail node object is located at the right of head node object and touching $(\text{DROF})^{-1} = \text{DLOF}$

Table 3.1 (continued)

DABOVE	12	tail node object is above head node object and touching $(DABOVE)^{-1} = DBELOW$
DBELOW	13	tail node object is below head node object and touching $(DBELOW)^{-1} = DABOVE$
DINB	14	tail node object is in the back of head node object and touching $(DINB)^{-1} = DINF$
DINF	15	tail node object is in front of head node object and touching $(DINF)^{-1} = DINB$

Table 3.2

Few Properties of the Relations

Relation	Symmetric	Asymmetric	Trans.	Intrans.
NEXT	1	0	0	1
CONTAIN	0	1	1	0
INSIDE	0	1	1	0
LOF	0	1	1	0
ROF	0	1	1	0
ABOVE	0	1	1	0
BELOW	0	1	1	0
INB	0	1	1	0
INF	0	1	1	0
DLOF	0	1	0	1
DROF	0	1	0	1
DABOVE	0	1	0	1
DBELOW	0	1	0	1
DINB	0	1	0	1
DINF	0	1	0	1

i.e.

$$(n,\text{NEXT},m_i) \in G \text{ where } m_i \in g_1 \text{ and } \forall\, m_j \in g_1 \quad j \neq i$$
$$(m_i,\text{INSIDE},m_j) \notin g_1$$
$$(m_j,\text{CONTAIN},m_i) \notin g_1 .$$

CONTAIN: This relation is an AND embedding type. Since this relation is transitive, not all the branches need to be present;

i.e. $\forall\, m_i$ in g_1

$$(n,\text{CONTAIN},m_i) \in G \mid (m_j,\text{CONTAIN},m_i) \in g_1 \text{ and } (n,\text{CONTAIN},m_j) \in G$$

or $$(m_i,\text{INSIDE},n) \in G \mid (m_i,\text{INSIDE},m_j) \in g_1 \text{ and } (m_j,\text{INSIDE},n) \in G$$
$$j \neq i \quad m_j \in g_1$$

We assume that every time a non-existing branch is implied by transitivity, it is created.

INSIDE: This relation is an OR embedding type;

i.e. $(n,\text{INSIDE},m_i) \in G$ where $m_i \in g_1$

is necessary and sufficient condition.

DABOVE:ABOVE: The node in question must be in "DABOVE"/"ABOVE" relation with a node inside the domain which is not contained in any node (object) of the domain. (Remember that neighborhood relations are defined only between brother nodes in a containment tree.) I.e.

$$(n,\text{DABOVE},m_i) \in G \quad \text{where} \quad m_i \in g_1 \text{ and } \forall\, m_j \in g_1 \quad j \neq i$$
$$(m_i,\text{INSIDE},m_j) \mid (m_j,\text{CONTAIN},m_i) \notin g_1$$
$$(n,\text{ABOVE},m_i) \in G \quad \text{where} \quad m_i \in g_1 \text{ and } \forall\, m_j \in g_1 \quad j \neq i$$
$$(m_i,\text{INSIDE},m_j) \text{ and } (m_j,\text{CONTAIN},m_i) \notin g_1 .$$

DBELOW:BELOW: The node in question must be in "DBELOW"/"BELOW" relation with a node inside the domain which is not contained in any node (object) of the domain; i.e.

$(n,DBELOW,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_j \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$

$(n,BELOW,m_i) \in G$ where $m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$.

DLOF:LOF: The node in question must be in relation "DLOF"/"LOF" to a node in the domain which is not contained in any node (object) of the domain; i.e.

$(n,DLOF,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$

$(n,LOF,m_i) \in G$ where $m_i \in g_1 \quad \forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$.

DROF:ROF: The node in question must be in relation "DROF"/"ROF" to a node in the domain which is not contained in any node (object) of the domain; i.e.

$(n,DROF,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$

$(n,ROF,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$.

DINB:INB: The node in question must be in relation "DINB"/"INB" to a node in the domain which is not contained in any node (object) of the domain.

$(n,DINB,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$

$(n,INB,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$.

DINF:INF: The node in question must be in relation "DINF"/"INF" to a node in the domain which is not contained in any node (object) of the domain.

$(n,DINF,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$

$(n,INF,m_i) \in G$ where $m_i \in g_1$ and $\forall\, m_i \in g_1 \quad j \neq i$

$(m_i,INSIDE,m_j) \mid (m_j,CONTAIN,m_i) \notin g_1$.

3.4.2 Other useful operations on this set of relations

There are some other useful operations on the set S of relations, i.e. $T_i \subseteq SxS$, where the set is closed under these operations. Tables 3.3 (a)-(d) defines three of these operations which physically correspond to rotating the scene 90°, 180°, 270°, respectively. As we see from Table 3.3(a), for each relation only one value is defined so T_1 is rather a function which maps the elements of S to some other elements of S (single value), so we have chosen functional representation for T_2 and T_3. It is easy also to see the following properties.

$$T_1 T_1 = T_2$$

$$T_1 T_2 = T_2 T_1 = T_3$$

$$T_1 T_1 T_1 = T_3 \ .$$

We make use of these operations to define certain transformations on general models to obtain different projection of the objects they represent.

Table 3.3(a)

T_1 (90° rotation)

	S \ S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NEXT	1	X														
CONTAIN	2		X													
INSIDE	3			X												
LOF	4								X							
ROF	5									X						
ABOVE	6						X									
BELOW	7							X								
INB	8					X										
INF	9				X											
DLOF	10														X	
DROF	11															X
DABOVE	12												X			
DBELOW	13													X		
DINB	14											X				
DINF	15										X					

Table 3.3(b)

Function T_1 (Rotation 90°)

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T_1(S)$	1	2	3	8	9	6	7	5	4	14	15	12	13	11	10

Table 3.3(c)

Function T_2 (Rotation 180°)

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T_2(S)$	1	2	3	5	4	6	7	9	8	11	10	12	13	15	14

Table 3.3(d)

Function T_3 (Rotation 270°)

S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T_3(S)$	1	2	3	9	8	6	7	4	5	15	14	12	13	10	11

4. MODELS AND PARSING

Communication between two different media occurs only through an intermediate medium interpretable by both. Meaningful oral conversation can occur between two human beings if each knows the language used by the other party. Otherwise through a translator the information is translated to an acceptable form to the listener. Written languages are only understood by the individuals who are able to transform it to internal forms of knowledge. An illiterate man will more or less receive the same information if a written paragraph is read to him, provided he is knowledgeable about the written fact. This suggests that the internal form of knowledge is more or less independent of the methods used in tapping this knowledge.

Of course we are not in any way suggesting that the growth of this knowledge itself is independent of the accessing methods used. On the contrary, we have learned from our experience with computers that the availability of higher level languages has made the formulation of problems otherwise extremely difficult, quite feasible.

Pictures are a universal means of conveying messages, since they are learned from natural environments which are more or less common to all human beings. A born infant before developing any linguistic capability is able to perform simple pattern recognition activities. The Chinese still use ideographs to convey messages (川 = river), and many natural languages retain ideographic elements even today.

The argument here is that knowledge should be represented in a hierarchical structure which can be easily modified and expanded. This structure should handle conceptual abstraction at all the levels, independent of irrelevant details.

A human statement about a fact is a collection of sentences which hopefully convey the intended meaning to the receiver. For example,

A boy is standing on the ground.

A ball is lying on the ground.

The boy is playing with the ball.

Here the objects can be either defined in context (their associations with other objects at the same level of abstraction) or in terms of their constituents (through the hierarchical structure of the knowledge). Graphical models are a valid approach in representing the knowledge. The facts relayed by the sentences in this example can be represented as in Fig. 4.1. The analysis is carried out by proving the theorem (goal) stated by each branch of this graph, using the body of knowledge present in the intelligent memory.

As any other program in a conventional programming language, scenes and pictures can be thought of as a collection of sentences, where each sentence describes an object. Relations between parts of different sentences can be translated into relations between the objects themselves. We also know that the pictures are context sensitive. For example, finding a human head and body in the scene does not necessarily mean that we have discovered a man, unless the head is connected to the body.

Even in string grammars where we have only simple adjacency relationship between the elements of the language. Grammars like Fig. 4.2(a) are context sensitive. (b) is a sentence in this language and (c) is the parsing sequence for this sentence. Rules like aB → ab can be applied only when "B" is preceded by "a", which establishes the required context for this replacement. In picture processing we are faced with several relationships between the picture primitives which make the contextual representation far more difficult than a simple adjacency relation. Relational graphs are an excellent method to represent this complex contextual information.

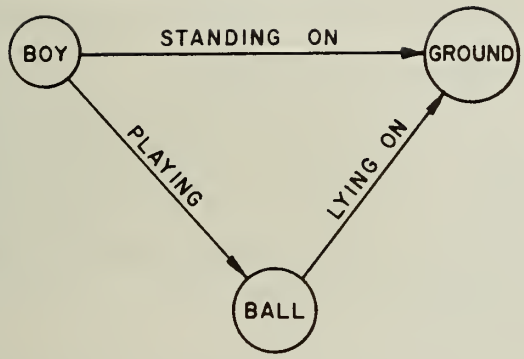


Fig. 4.1. Graphical representation of a collection of English sentences.

g: S → aSBC | aBC bB → bb
CB → BC bC → bc
aB → ab cC → cc

(a) grammar

(b) aabbcc

s → aSBC →

aaBCBC → aabCBC →

aabBCC → aabbCC →

aabbcC → aabbcc

(c) parsing of the sentence (b)

Fig. 4.2. A context sensitive language and parsing.

With relational-graph representation of the rules in a grammar, we have to depart from conventional parsing methods and use graph transformational techniques. In Fig. 4.3 we have shown how a relational-graph can be used to represent the structure (or context) of a table.

4.1 Graph Structure Definitions

Here we define a graph structure, which corresponds to a production system in conventional string grammars. Let

$N = \{n_1, n_2, \cdots, n_m, \lambda\}$ be the set of all nodes in the graph structure,

where λ is a null node.

There is a partial ordering between the nodes of a graph structure. In this partial ordering each node is immediately above all the nodes which participate in its structural definition.

If n_j is immediately above n_i (in other words $n_j > n_i$) then, n_i is said to be an immediate predecessor of n_j in the partial ordering.

Definition: A primitive node is defined as a node with an empty list of predecessors. λ is a primitive node. All of the nonprimitive nodes are called super nodes.

Imposing the following restrictions on the partial ordering of nodes, and introducing structural bonds by labeled branches, we can define the graphs of our system as follows: Let

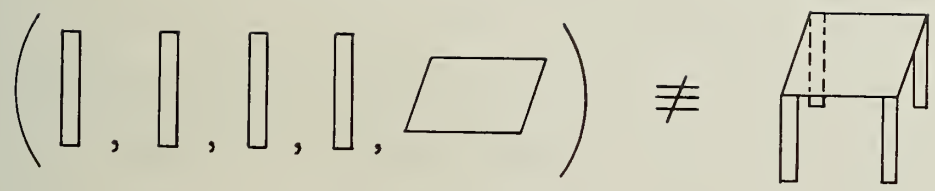
N_i be the set of nodes immediately below n_i,

N_j be the set of nodes immediately below n_j,

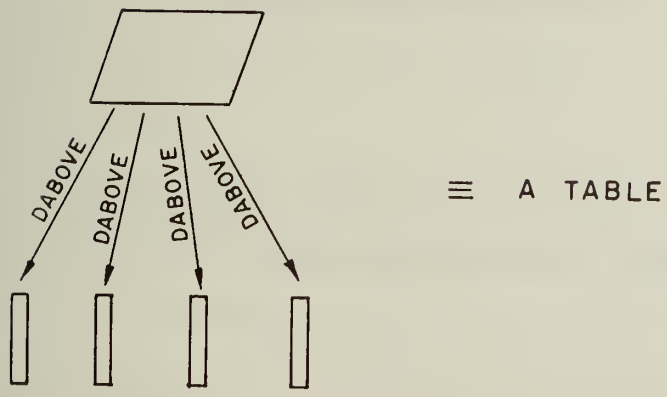
then we impose that

either $N_i \cap N_j = \text{null}$

or $N_i = N_j$ where $N_i \cap N_j = N_i = N_j$.



(a) parts of an object



(b) structural representation of the object

Fig. 4.3. Context sensitivity of pictures.

The above equivalence relation will divide the set of nodes to the equivalence classes. Each class is specified with a set of nodes which are immediate predecessors to all the nodes in this class. We attribute a graph g to each class as follows:

$$g_\ell = (N_i, B_i),$$

where $N_i \in N$ and it is the set of predecessors to this class;

$B_i \in B$ where for every branch

$$b_k = (n_i, u_p, n_j) \quad (u_p \in U),$$

if $b_k \in B_i$ then $n_i, n_j \in N_i$. Here,

$B = \{b_1, b_2, \cdots, b_p\}$ set of all the branches in the structure,

$U = \{u_1, u_2, \cdots, u_k\}$ set of labels for branches,

so, $B = NxUxN$.

Now assume that

$G = \{g_1, g_2, \cdots, g_q\}$ set of all the graphs in the structure.

Among these graphs besides the set defined above there are the graphs which are formed by the nodes which do not have any successor in the partial ordering of the nodes. We call this set of graphs as FGS (final graph set). From the above definitions it is clear that each super node has a graph associated with it. The set of graphs form a partial ordering as follows:

Graph partial ordering: A graph g_i is immediately above g_j if and only if g_j is associated with one of its super nodes. We call this partial ordering σ.

With this preparation we now define our graph structure GS as follows;

$$GS = (N, B, G, \sigma|U, A).$$

Here again,

$N = \{n_1, n_2, \cdots, n_m, \lambda\}$ set of all nodes.

n_i is a super node, if it has a graph associated with it.

n_i is a primitive node if it has no graph association.

λ is a null node.

$U = \{u_1, u_2, \cdots, u_k\}$ set of labels for branches

$B = \{b_1, b_2, \cdots, b_p\}$ set of branches in the structure

where

$b_i = (n_i, u_\ell, n_j)$ and $u_\ell \in U$, $n_i, n_j \in N$

so,

$B \subseteq NxUxN.$

$G = \{g_1, g_2, \cdots, g_q\}$ set of all the graphs in the structure (FGS and the set associated with the super nodes),

σ is a partial ordering between graphs,

$A = NA \cup BA \cup GA$

$= \{a_1, a_2, \cdots, a_r\},$

is a set of attributes for nodes, branches and graphs. Each a_i is a function such that,

$a_i: \{N|B|G\} \rightarrow v_i$

where v_i is a set of values. For example, $a_i(g_k \cdot nodel) = g_j$ is interpreted as: graph g_j is associated with the node "nodel of graph g_k". Equivalently, we may consider the branches as a set of relations $\{Ru_1, Ru_2, \cdots, Ru_r\} = \{R_u\}$ where,

$R_u \subseteq NxN,$

and a branch $b_i = (n_j, u_k, n_\ell)$ exist if and only if $(n_j, n_\ell) \in Ru_k$.

By having the graphs to form a partially ordered set, not only we can economize the space used for modeling our universe, but also we have an implied hierarchy built into the system, which assists us in search of domains to apply transformations.

Once again to put everything in prospective, in our picture analysis scheme, a graph structure constitutes our body of knowledge about the universe. Nodes represent picture primitives or higher level composite objects. Labeled branches represent binary relations between parts of objects (syntactical context). Graphical models of the objects represent the local domain or context of a transformation. As we have seen in chapter 3, the range of transformation is always a single node. The attributes carry most of the semantical information of any particular application. Along with the transformation there are semantical procedures which decide the combined semantical information to be carried to the formed composites.

An example would be helpful to understand the above material.

$N = \{n_1, n_2, n_3, \cdots, n_{10}\}$ set of nodes.

$n_1, n_2, n_3, n_4, n_5, n_6, n_9$ are primitive nodes,

n_7 is a successor to n_5 and n_6,

n_8 is a successor to n_1, n_2, n_3, and n_4,

n_{10} is a successor to n_5 and n_6.

The partial ordering is shown in Fig. 4.4(a).

$G = \{g_1, g_2, g_3, g_4\}$

g_1 is associated with super node n_8.

g_2 is associated with super nodes n_7 and n_{10}.

$g_3, g_4 \in$ FGS, i.e. the Final Graph Set.

$U = \{\text{above}, \text{next}\}$ set of branch labels;

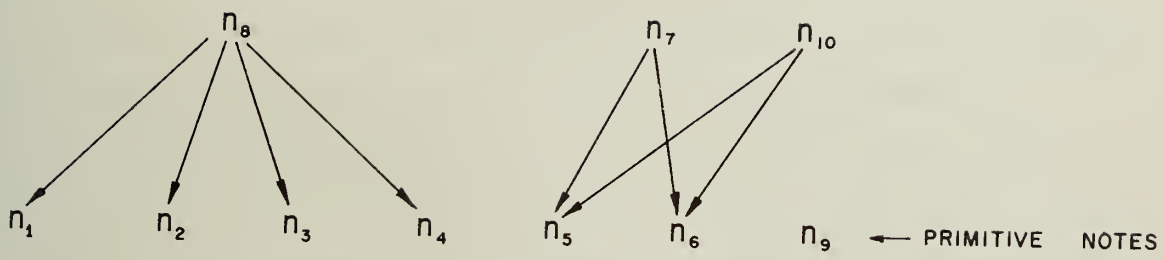
introducing the branches, each graph is defined as follows:

$g_1 = \{n_1, n_2, n_3, n_4, (n_1, \text{next}, n_2), (n_2, \text{above}, n_3), (n_2, \text{above}, n_4)\}$,

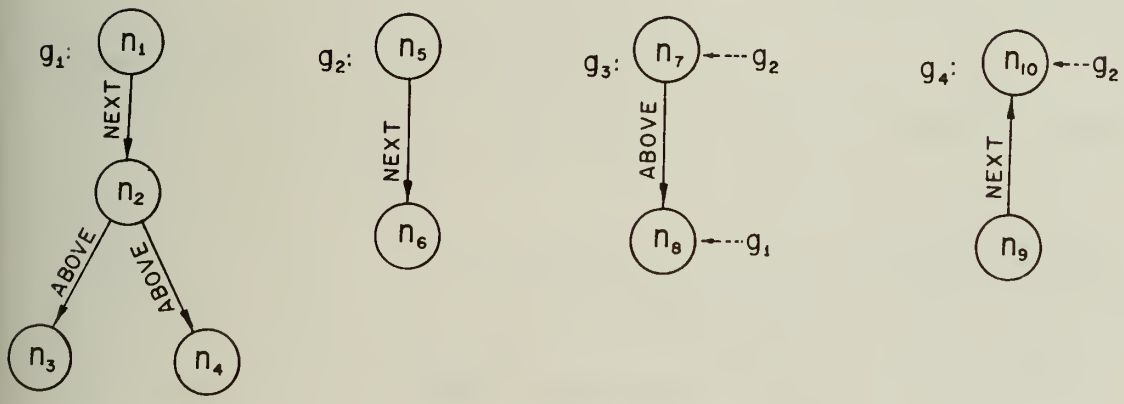
$g_2 = \{n_5, n_6, (n_5, \text{next}, n_6)\}$,

$g_3 = \{n_7, n_8, (n_7, \text{above}, n_8)\}$,

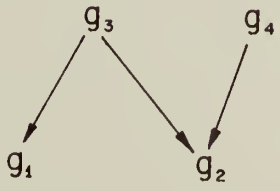
$g_4 = \{n_9, n_{10}, (n_9, \text{next}, n_{10})\}$,



(a) partial ordering between nodes



(b) graphs of the system



(c) partial ordering σ

Fig. 4.4. An example of a graph structure.

and σ is the partial ordering 4.4(c) of the graph set G.

$$A = \{GRAPH, \cdots\}.$$

The node attribute "GRAPH" is used to associate graphs with the super nodes.

$$GRAPH(n_8) = g_1$$

$$GRAPH(n_7) = g_2$$

$$GRAPH(n_{10}) = g_2.$$

Since there is no need for all nodes to have unique names, qualification of node references can be necessary.

$$g_1.n_2,\ g_2.n_5,\ \text{etc.}$$

But, all the nodes in the same graph must have unique names. Here we give a few examples of SOL functions:

$$NOF(g_1) = \{n_1, n_2, n_3, n_4\}$$

$$BOF(g_2) = \{(n_5, next, n_6)\}$$

$$INCBR(g_3.n_8) = \{(n_7, above, n_8)\}$$

$$OUTBR(g_1.n_2) = \{(n_2, above, n_3), (n_2, above, n_4)\}$$

$$ADJBR(g_1.n_2) = \{(n_1, next, n_2), (n_2, above, n_3), (n_2, above, n_4)\}.$$

4.2 Parsing the Graphical Representation of the Scene

In string languages parsing is accomplished either in bottom-up or top-down methods. Top-down parsing is goal oriented, where we set up goals and try to reach them through a set of subgoals in the parsing tree. When a set of subgoals fail, we try an alternative set of subgoals, and if all the subgoals, achieving the same goal, have failed we have to set up a new goal and proceed. In bottom-up parsing we depend on local evidence. When enough evidence, which satisfies a subgoal phrase is at hand, we will parse this evidence as that subgoal and proceed. If an accomplished subgoal fails to produce any higher level goals, then it is an erroneous conclusion and we

have to backtrack. Our method of parsing is similar to the bottom-up parsing, but here instead of symbols, graphs are pushed into our parsing stack. We also can use top-down parsing when heuristically it is beneficial to do so. Before discussing our parsing method we make the following definitions. As we have mentioned before, each super node in our graph structure has a graph associated with it.

Definitions: A son-set of a graph is a set of graphs which are associated with its super nodes.

A father-set of a graph is a set of graphs which have this graph associated with one of their super nodes.

Primitive-graph set (PGS) is the set of graphs in which all of their nodes are primitive.

Now we define our parsing graph as follows: The parsing graph is a graph which has the same number of nodes as the number of graphs in our graph structure. Each node has one graph associated with it. A set of weighed branches leaves each node which arrives at nodes whose associated graphs are the sons to the graph associated with this node. Since there is a one-to-one correspondence between the nodes in this graph, and the graphs in the graph structure, we use them interchangeably.

There are several advantages in representing our parsing tree in a graphical form

(a) The dynamic nature of a graph makes modification very easy—which is essential in any system with learning capability. For example, addition and deletion of graphical rules to the graph structure is straightforward.

(b) Through association of variables with branches we can affect the parsing order.

(c) We can use SOL to manipulate this graph.

Fig. 4.5 shows the structure of this graph. OUTBR(node) specifies the set of branches whose head nodes form the son-set of this node. INCBR (node) specifies the set of branches whose tail nodes form the father-set of this node.

As we have seen in chapter 2, scenes are represented as graphs whose nodes correspond to primitive regions of the pictures and branches represent binary relations between these regions. Node attributes (shape, color, texture, size, etc.) carry semantic information about each node.

The purpose of parsing is to find the set of recognizable objects in the scene which have a graphical model in our graph structure (universe). Fig. 4.6 shows the dynamic parsing tree which has its root at the attention-point (AP) and propagates to higher level objects. Parsing procedure for each object is complete when the recognized graph belongs to final graph set (FGS) or it is a meaningful object and none of its fathers can be recognized. When all the fathers of a recognized rule have failed and this rule does not represent a meaningful object in our universe, the inverse transformation to this rule is applied to the scene and will restore the environment to the preapplication of this rule. Now, the next rule in the ordered set at the previous level will be tried.

Fig. 4.7 shows the flow chart of the basic recognizer. Here, we have avoided the details of each sub-function which are described in the following subsections. There are also other easily implementable concepts which we will introduce later on in this chapter and in chapter 6.

a. Best-Match Concept: This enables us to recognize incomplete or partially hidden objects in the scene.

b. Scoring Techniques: This will enhance the performance of the system in a repetitious environment.

Parsing graph:

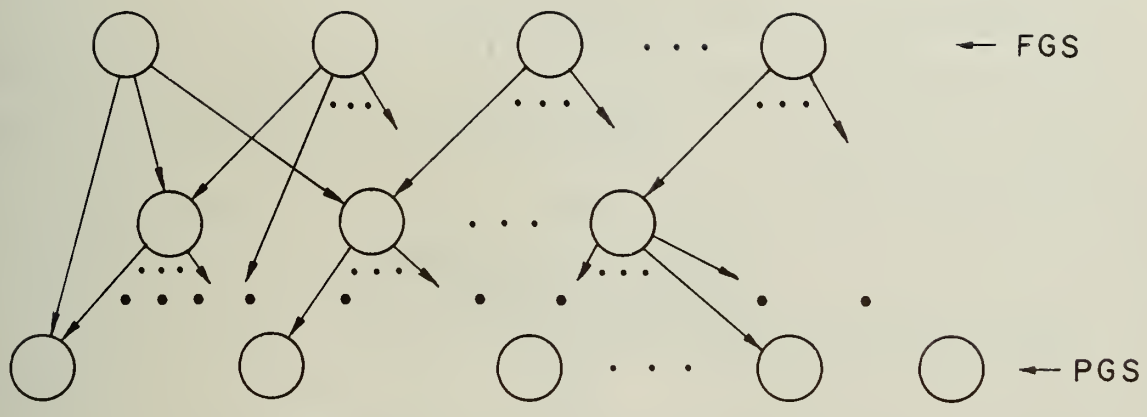


Fig. 4.5. The graphical representation of parsing graph.

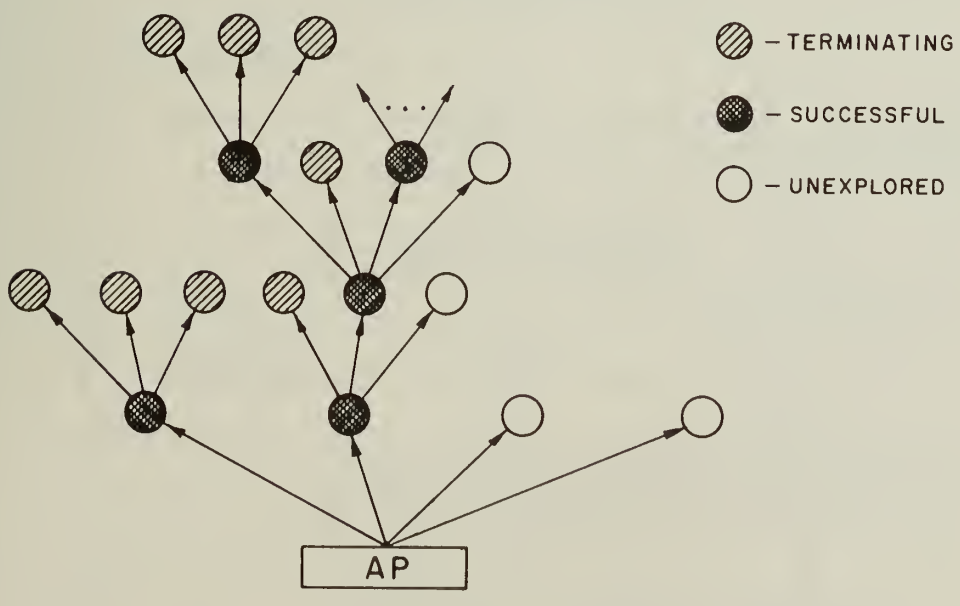


Fig. 4.6. Dynamic parsing tree.

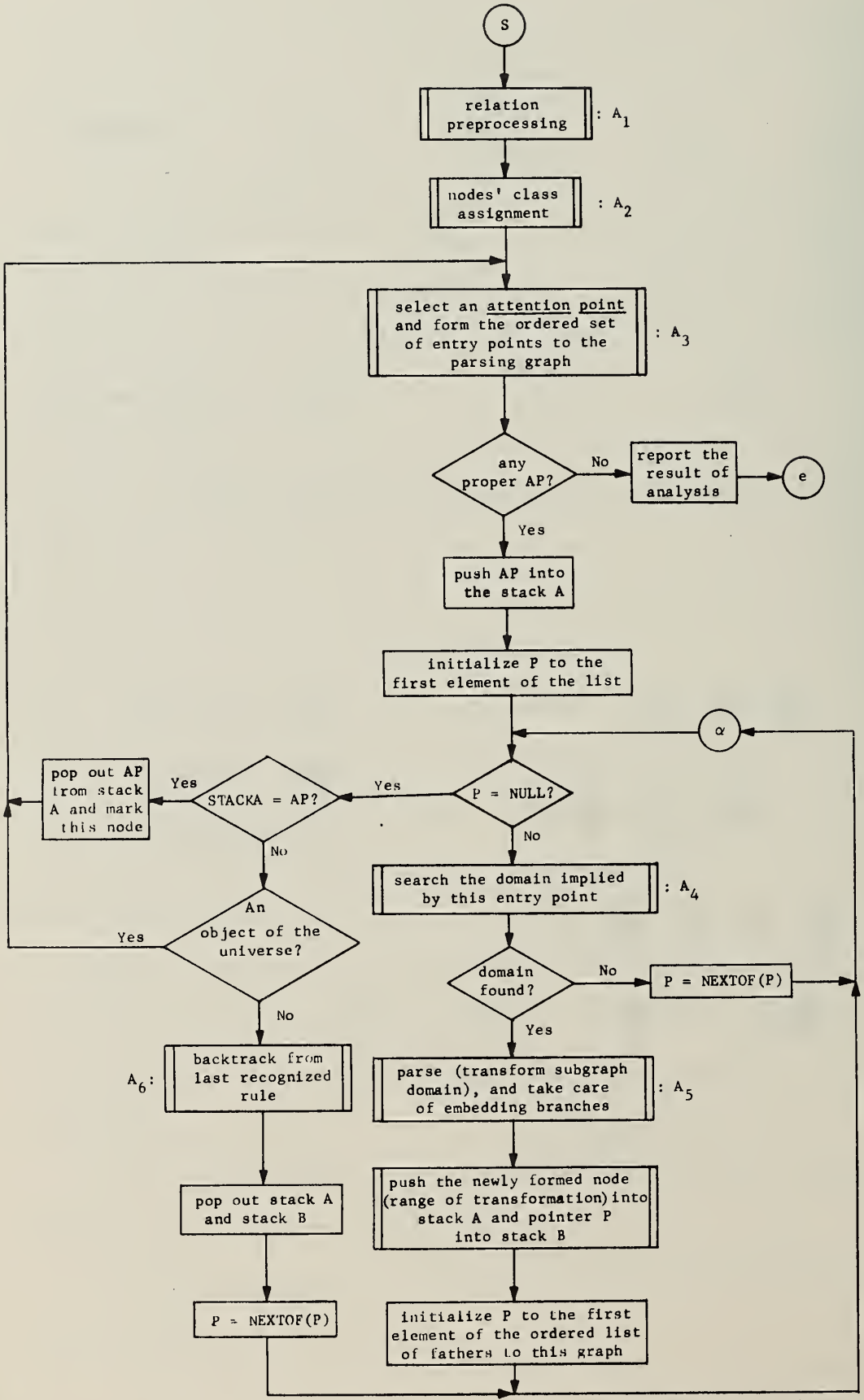


Fig. 4.7. Flow chart of basic recognizer.

c. Heuristic Techniques: These techniques make use of more global information to enhance the performance of our system.

d. Learning Capabilities: These are the functions which enable our system to add or combine objects in the universe and also enhance the performance of recognizer. We discuss these aspects of recognition in chapter 6.

Our recognizer functions mainly in two modes:

1. Learning mode: saves all the useful acquired information from this experience for further use in future.

2. Non-learning mode: The extent of information saving is negligible compared to learning mode.

4.2.1 Relation preprocessing

Exploiting certain properties of the relations used in representation of our universe, we can substantially reduce the number of relations or facilitate further manipulation.

The class of relations we described in chapter 3 is closed under inversion. We can describe our graph structure by using only relation r instead of r and r^{-1}. In the graphical input of the scene the branches with label r^{-1} are labeled as r and the direction of the branch is reversed. This preserves the syntactic and semantic implications of the scene while greatly reducing the recognition effort. Other properties of these relations like transitivity can be taken care of at the preprocessing level (creating all implied branches, or creating these as they are needed in the course of action). These functions are performed in action box A_1 of Fig.4.7.

4.2.2 Nodal class assignment

Nodes in the original scene are picture primitives and must be assigned to one or more primitive nodes of our graph structure. This assignment is

carried out through the classification of attribute values assigned to each primitive node. For continuous attribute values the number of possible physical nodes (primitive regions) are infinite, but by dividing the continuous values of the attributes into discrete intervals, we will partition the infinite set of primitive nodes to finite set of primitive classes.

For each attribute value of a primitive we will assign it to the class of primitives whose value interval for this attribute includes the specified value. If for some attribute value there is no specified class we mark this node as noise.

definition: Two primitive nodes are semantically equivalent if and only if for all the specified attribute values they have the same class assignment.

This function is shown in action box A_2 of Fig. 4.7.

4.2.3 Selection of an attention point

In this section we will describe the procedure which discovers the set of feasible rules (entry points to the parsing graph) to be tried, starting at any primitive region which we call an attention point (AP).

It is clear that each primitive class will indicate a set of primitive nodes in our graph structure, and these primitive nodes in turn will imply a set of graphical rules (entry points into the parsing graph) which have one of these primitives as their nodes. So with each primitive class we will have a set (possibly ordered) of entry points associated with it.

For each node of the scene, selected as an AP, we follow the following procedure:

S: set of entry points

a. $S = \phi$ (null set)

b. For each specified attribute value class C_i, assume S_i is the set of associated entry points.

IF $S = \phi$ THEN $S = S_i$

ELSE $S = S \cap S_i$

If $S = \phi$ THEN mark this node as noise and pick another AP and go to a. If this is not possible the recognition is complete.

c. Repeat b. for next attribute. If all the specified attributes have been tried then go to d.

d. Test the acceptability of this AP (for example CARD(S) must be reasonably small). If it is acceptable then leave this routine and proceed. Otherwise, use some heuristics to improve upon this.

One of the possible heuristics is to pick up a structurally tied neighboring node, and develop the set of entry points for this AP. Then by intersecting these two sets, we can find a smaller set of entry points.

Action box A_3 in Fig. 4.7 refers to this procedure.

4.2.4 Search of domains for transformations

The implied entry point in the parsing graph will call for the search of the domain corresponding to its associated graph. The algorithm of this search is given below, but to understand this algorithm let us make a few points clear.

Definition: Two super nodes are semantically equivalent if and only if they have the same graphical rule associated with them.

The search of domain is a recursive procedure since the super nodes require that their corresponding transformations be applied to the scene. The parsing procedure will be called when a domain corresponding to a super node is found. The backup procedure will undo the parsing when a wrong

decision has been made. This can happen in one of the three following ways:

1. A wrong branch correspondence has been made.
2. The embedding rules have eliminated a necessary branch.
3. The match is not complete and we have to select another equivalence in the graphical rule for the starting point.

Outline of the Algorithm Used for Domain Discovery

1. Find the set of all nodes in the graphical rule which are semantically equivalent to the starting node in the scene.
2. If the list is exhausted return the best match. (There should be enough information in the best match to enable us to reparse the back-tracked rules.) Otherwise pick up the next element (E_1) of the list of equivalences as a match for the starting node (E_2).
3. Form the ordered list of the ADJACENT branches to this equivalent node, (call it S_1) and to E_2, (call it S_2). Initialize pointer P_1 and P_2 to S_1 and S_2, respectively.
4. Find a branch in S_2, starting at P_2, which is equivalent to the branch pointed to by P_1. If the branch, which is not already matched with another branch in the rule, exists then go to 5. Otherwise: no match can be found for the branch pointed to by P_1.

   Put, P_1 = NEXTOF(P_1) and initialize P_2 to the set S_2.

   Go to 4.
5. Compare the end nodes of the branches pointed to by P_1 and P_2. If they are equivalent reflect this fact into the MERIT of the current match. [Note: If the end node in the graphical rule is a super node, the discovery of its equivalence in the scene will involve the recursive call to this routine; and besides, we have to check that the branch P_2 leading

to this match will remain valid after parsing. If P_2 was eliminated as the result of parsing we have to back up from this newly formed super node,

put P_2 = NEXTOF(P_2) and go to 4.]

P_1 = NEXTOF(P_1).

Mark these branches (P_1 and P_2) and their end nodes.

If P_1 = NULL then look for a matched node in the rule which has not yet been picked as the starting point. If such a node exists replace E_1 with that and E_2 with its equivalence in the scene; go to 3.

Otherwise, test for completion of the search. If search is complete then return the set of matched node, representing the domain.

Otherwise, set E_1 to the initial starting node, back up from recognized super nodes (except the one that may correspond to starting node), save the best match, and go to 2.

Otherwise, go to 4.

Otherwise, put P_2 = NEXTOF(P_2) and go to 4.

This procedure corresponds to action box A_4 of the Fig. 4.7. More detailed explanation of these algorithms are case dependent. We will clarify these in connection with our example universe in chapter 5. We have also to point out that the above algorithm does not exhaust all the possibilities (like branch and node may both have a match, while it is still a wrong match), but we know that they seldom happen;and even if they happened, they will change the context which eventually will force us to back up and correct them.

4.2.5 Actual parsing of a found domain

As we mentioned in chapter 3, in our class of transformations the range of transformation is a single node, and a single parsing routine can perform

all the transformations as follows:

a. create a super node,

b. associate the graphical rule which caused this transformation with this node,

c. take care of embedding branches.

When the parsing procedure is applied, the nodes and branches of the domain will become temporarily inactive. As far as further processing is concerned this is equivalent to removing these elements from the scene graph. The branches cutting the domain are also temporarily made inactive; but if our embedding rules dictate a replacement, new branches are embedded between the nodes outside and the newly created node.

4.2.6 Back-up procedure

Back-up procedure performs the inverse action of the parsing procedure. The super node and all its adjacent branches become inactive. All the nodes and branches of its domain and also the branches cutting into the domain will once again become active elements of the scene graph.

When this procedure is called within A_4, it will be called recursively until no super node is left in the domain of the back tracked super node. Since some of the elements in the best match set might point out to these back tracked super nodes, we do not eliminate their memories when we make them inactive. Then, it is possible to transform their domains back to this level of parsing without repetitive and extensive search for the domain which is already found. This is very useful in connection with the best match feature of our recognizer.

4.2.7 Heuristics

Heuristics are generally techniques which avoid exhaustive search and take a short path to the goal. There are several heuristics which are very

useful in speeding up the parsing procedure.

1. Assignment of values to the variables associated with the branches in the parsing graph. These values can affect the order in which the successors are tried. The recognizer can change these values dynamically through experience.
2. We can also employ a scoring system which can affect the ordering of the list of entry points associated with each primitive class.
3. At any point of search we can pick up another node besides the one under consideration which has a structural tie to the later node. Then instead of trying the set of successors to each node, we may try only the intersection set of these two sets. We can try this procedure repeatedly until we are satisfied.

   If we follow the following procedure in forming the intersection of two ordered set, we will preserve the ordering.

   $S = S_1 \cap S_2$

   a. Mark all the elements of S_1.
   b. Scan the elements of S_2 in order, and include it in S (at the bottom) if it is marked.
4. There are some semantical associations between the objects which occur in the same scene. When the presence of an object is highly probable (like chair and table), we can form a set of all its descendent rules and itself. Then, we will intersect this set with the set of entry points inferred from the current node (i.e. AP), and reorder the elements in the later set such that the elements of the intersection set will be tried first.
5. In the recursive call of the search for domain (looking for sub-domains), less than complete match is normally acceptable.

4.3 Best-Match Feature of the Recognizer

In the real world, the objects do not always appear in the same way they are specified in the models. They can be occluded or partially hidden from the view. Our basic recognizer can be easily modified to enable it to handle this partial matching.

Definition: Figure of Merit (FOM) is a variable which defines the degree of partial matching. Its value ranges from 0 to 1, where 0 specifies no matching, while 1 is for complete matching.

Let us define the recursive function EXPAND as follows:

EXPAND: Primitive node → Primitive node

: Super node → EXPAND {set of nodes in its domain}.

And it has the following property,

EXPAND {a,b} = EXPAND(a), EXPAND(b).

If S_1 is the set of nodes in the domain and S_2 the set of nodes in the graph currently being matched, then

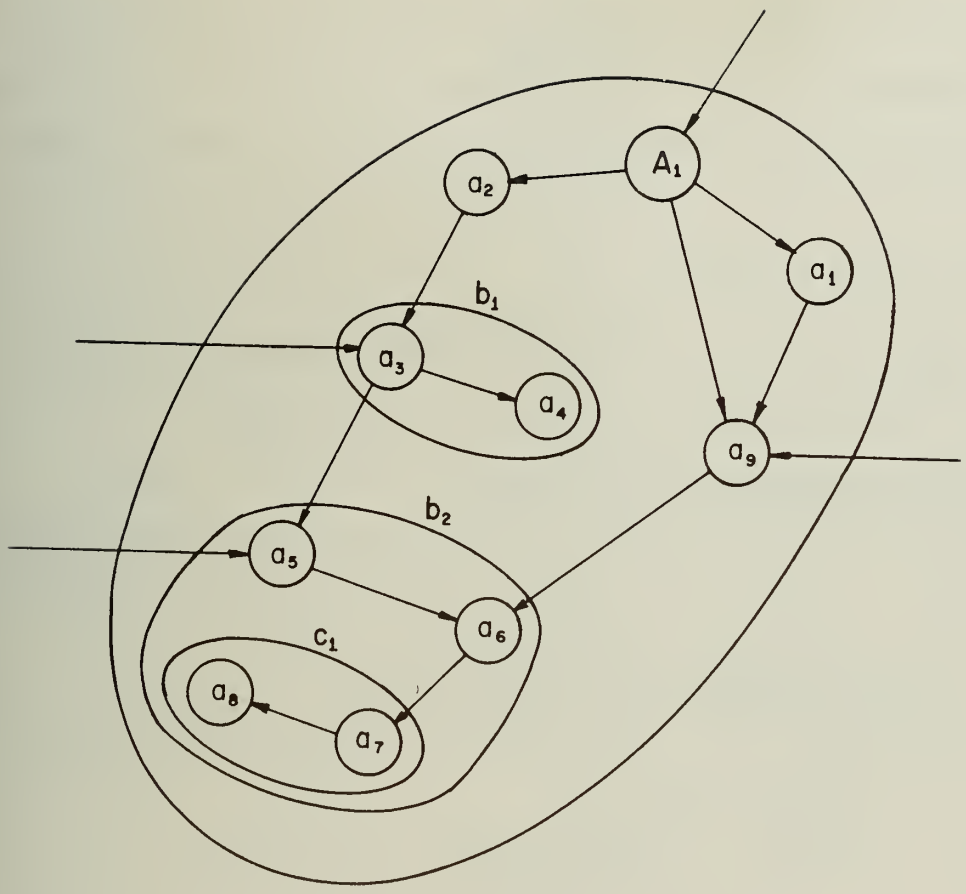
$$FOM = \frac{|EXPAND(S_1)|}{|EXPAND(S_2)|},$$

is our definition for figure of merit. In Fig. 4.8(a) we have shown an example of a domain. In this example,

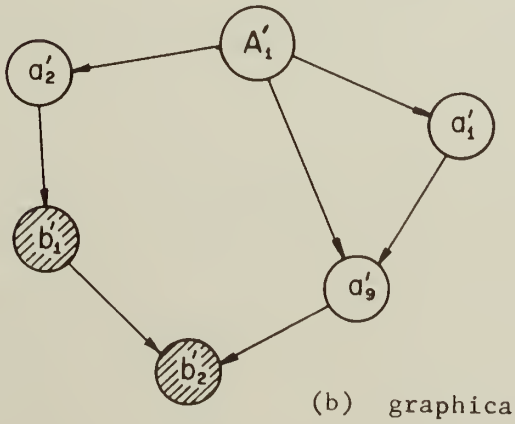
$$S_1 = \{A_1, a_1, a_2, a_9, b_1, b_2\}.$$

$$\begin{aligned} EXPAND(S_1) &= EXPAND\{A_1, a_1, a_2, a_9, b_1, b_2\} \\ &= EXPAND(A_1), a_1, a_2, a_9, EXPAND\{a_3, a_4\}, EXPAND\{a_5, a_6, c_1\} \\ &= EXPAND(A_1), a_1, a_2, a_9, a_3, a_4, a_5, a_6, EXPAND(a_7, a_8) \\ &= EXPAND(A_1), a_1, a_2, a_9, a_3, a_4, a_5, a_6, a_7, a_8 \ . \end{aligned}$$

Since the cardinality of the EXPAND for each super node in the graph structure is a constant, we can have this associated with the nodes in the



(a) maximal domain matching the graphical rule.



(b) graphical rule being matched

Fig. 4.8. An example of a graphical rule and its corresponding domain in the scene.

parsing graph. This constant is helpful in the sense that we can find the $|EXPAND(S)|$, by having the "FOM" for its domain. For example if A_1 (starting point) was a super node and its associated graph g_i had the constant "$CONS_i$" then,

$$|EXPAND(A_1)| = FOM_{A_1} \times CONS_i \ .$$

definitions: MFOM (Minimal Figure of Merit) is the minimum value for FOM when we consider that any partial matching has occurred at all.

AFOM (Acceptable Figure of Merit) is the level of partial matching which we accept as a match when the algorithm of domain search is called recursively. The domain is actually parsed to a super node, only and only if this level of partial matching is achieved.

We can calculate the "FOM", (Figure of Merit for domain D) as follows:

$$FOM_D = \frac{N_D + \sum_i^i FOM_i \times CONS_i}{CONS_D} \ .$$

Here,

N_D: number of primitive nodes in the domain,

i: index for a super node in the domain,

FOM_i: figure of merit for super node i,

$CONS_i$: constant associated with the graphical rule matching with the domain of this super node,

$CONS_D$: constant associated with the graphical rule being matched.

In our recognition algorithm, at any stage of search there are a set of possible successors that can be tried. Fig. 4.9 shows the algorithm of search for the best match at any one stage. In this flow chart the variables have the following interpretations:

AP: attention point, which is the result of the last stage of search (can be either primitive or super node),

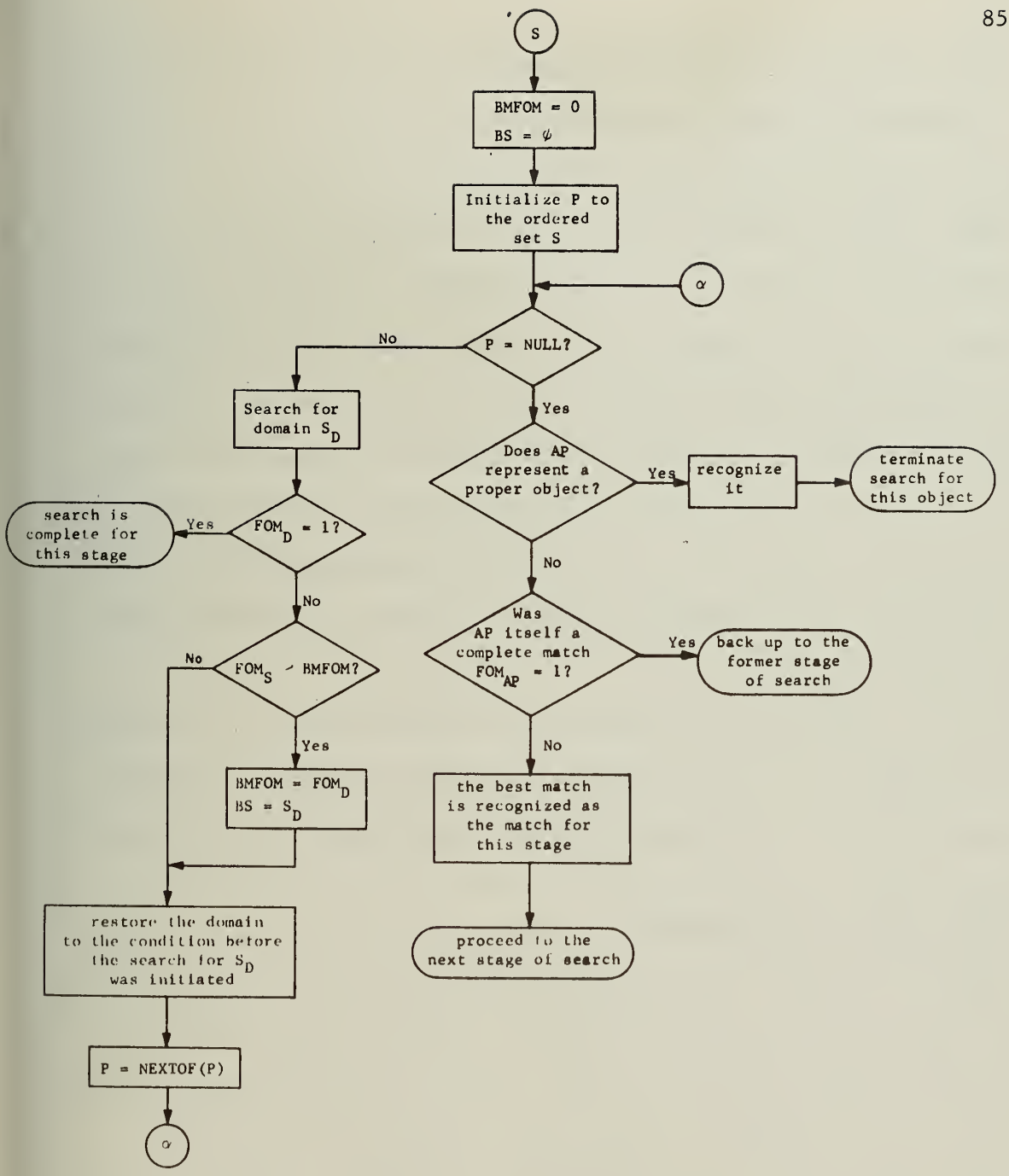


Fig. 4.9. Flow chart of the best match algorithm.

BMFOM: figure of merit for the best match,

S: ordered set of successors to AP,

S_D: domain of the current alternative (set of nodes),

FOM_D: figure of merit for the current alternative,

BS: set of nodes in the best match domain.

When our task is to pick the best of some alternative, it is obvious that the conditions must be exactly the same before we try each alternative. This is accomplished by recursive calls to the back-up procedure for each super node (excluding AP) in the current domain before the next alternative is tried. But to avoid repetitive search for the best match, the structure of higherarchical clustering in its domain is saved to be reparsed later on without actual searching.

In the domain search, when we have more than one node in the graphical rule which matches with AP, we are faced with the same task of selecting the best match. This is accomplished in the same manner as in the recognizer.

4.4 Other Useful Transformations

There are several useful operations that can be applied to the scenes as well as to our universe which expand the ability of the recognizer to recognize the objects in a more realistic world. These operations can be thought of as transformations which conform the scenes to our universe or modify the universe to have the knowledge transformed to a form which helps the scene interpretation.

Among this class of transformations are the ones which can be applied to images of cerebral context to discover deformity. In matching the cross sections of a brain to the brain atlas, some of the region parameters or relation structures could be deviant from the atlas, and by defining a

transformation which makes the matching possible, we will have discovered the deformity which in turn will tell us about its effects on different sections. Guzman was the pioneer [19] in discovering a simple set of rules which can be used to transform the scenes into their constituent 3-dimensional bodies. He mainly uses the vertices type properties to merge their associated faces. This can be thought of as a preprocessor to our 3-dimensional, model-based, parsing scheme. We have used some of the simple rules (T-joint, Y-joint, etc.) to demonstrate its compatibility with our modeling system, and the results will be discussed in chapter 5.

4.4.1 Rotational transformations

A graphical model of an object contains all the pictorial knowledge about that object. As a matter of fact, the relational description of the knowledge about the universe should be flexible enough to allow us to reflect modifications caused by an external or internal factor. For example, in sociology hate-like relationships could be described in a relational graph. Now if the effect of external factors like heat or cold or internal factors like death and marriage are known, then we should be able to transform the body of knowledge to arrive at the new relationships. Here in connection with picture processing we define the rotational transformations which from the unique models of the objects will construct the models for the same objects in different orientations. In chapter 3 we defined a set of operations (i.e. T_1, T_2, T_3) and found out that the class of relations we chose were closed under these operations. Using these operations now we can define the following rotational transformations: T_{R90}, T_{R180}, T_{R270}.

1. For T_{R90}, T_{R180}, and T_{R270} apply the operations T_1, T_2 and T_3 respectively to the set of relations in the graphical model.

2. Semantics of the nodes representing object's primitive regions are modified to represent their projections at this angle.
3. Semantics of the super nodes are unchanged.
4. Eliminate the parts which are now hidden from our view because of the new orientation. We will have the relative size of each part associated with the node representing it.

Now if one of the branches in the graph tells us that a smaller part is located directly behind a bigger part, its corresponding node should be mapped to a null node by this transformation (in other words, deleted from the graphical representation). Or if it will be partially hidden, this fact should be reflected in the semantics of the node representing it. For example, the parts of a human face will disappear from the view by a 180° rotation. In chapter 5 we will show the results of our experimentation with a few 3-dimensional models using these transformations. Since these transformations are irreversible (deletion), we have to use a separate copy of the graphical rule for each transformation.

5. APPLICATION

We have developed a general methodology for global picture analysis by graph transformations. In this chapter we show the results of applying this methodology to a simple class of cartoons. We have avoided mentioning the detailed algorithms since they are the same procedures outlined in chapter 4. In this application we have also tried to conform as much as possible to the general procedural steps discussed in the last chapter. Although we have made this application quite simple for reasons of clarity, it involves all the basic factors and complexity of the general case. We assume the objects in the scene are well formed, and for the time being non-occluded.

5.1 Primitive Classes

We have learned that the nodes in the graphical representation of input picture represent the picture's regions, primitive nodes in the model graphs represent the primitive regions (or a set of alternative primitive regions) in the model, and super nodes represent higher level objects (a graph in our graph structure). We also know that the semantic equivalence of two nodes is defined as follows:

> Two primitive nodes are semantically equivalent, if for all of their specified attributes they belong to the same primitive classes. Two super nodes are semantically equivalent if they have the same graph associated with both.

Now we have to define our primitive nodes' attributes which divide our primitive nodes into primitive classes, such that a set of these classes, one for each attribute, define a primitive region.

5.1.1 Shape attribute

In Table 5.1 we have shown the class of prototype shapes which divide our primitive regions into 29 equivalent classes. Methods discussed in [3] by Maruyama can be used to assign each primitive region to one of these classes (the closest in shape). When no shape assignment is possible we assign this primitive to the class of primitives which have an undefined shape (#30).

5.1.2 Compactness attribute

This attribute can actually be a feature of the shape attribute, but has been brought up to emphasize its relative importance and also to have more than one attribute for the primitive regions. This attribute is useful in that it divides the primitives into classes of narrow and broad regions. It is defined as follows:

$$L = 16(1-2\sqrt{\pi A}/P)+ 1,$$

where A is the area and P is the perimeter of the region. The constant 16, besides the fact that we found it appropriate for this particular application, has no general significance. This attribute divides our primitives to 7 different distinct classes as follow:

class #1: $1 \leq L < 3$

class #2: $3 \leq L < 6$

class #3: $6 \leq L < 8$

class #4: $8 \leq L < 10$

class #5: $10 \leq L < 12$

class #6: $12 \leq L < 15$

class #7: $16 \leq L$.

The value of attribute L is 1 for a circle and 17 for any region with 0 area.

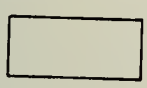
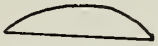
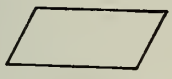
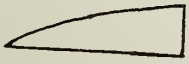
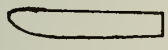
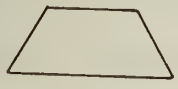
No.	SHAPE	No.	SHAPE	No.	SHAPE
1		11		21	
2		12		22	
3		13		23	
4		14		24	
5		15		25	
6		16		26	
7		17		27	
8		18		28	
9		19		29	
10		20		30	Undefined

Table 5.1. Table of the values for shape attribute.

5.2 Relations

We have chosen to use the same class of relations as those given in chapter 3. There we found that this class is closed under inversion.

$(NEXT)^{-1} = NEXT$

$(CONTAIN)^{-1} = INSIDE$

$(DROF)^{-1} = DLOF$

$(DBELOW)^{-1} = DABOVE$

$(DINB)^{-1} = DINF$

$(ROF)^{-1} = LOF$

$(BELOW)^{-1} = ABOVE$

$(INB)^{-1} = INF$

Although we allow the user (or preprocessor) to use all of these relations in generating the graphical representation of the scene, we can use a simple preprocessing procedure which changes all the branches labeled by left hand side of the above equation to labels on the right hand side and change the orientation of the branch. This will allow us to use only the set of right hand side relations. Of these relations NEXT, INSIDE, DLOF, DABOVE, and DINF are called structural relations, which are used in defining the graphs of objects in the universe. Since in our graphical analysis we are mainly concerned with these structural relations, we have chosen to use another preprocessor to generate the implied branches by transitivity. Among the structural relations only "INSIDE" is transitive, so in our input graph if we have branches (A,INSIDE,B) and (B,INSIDE,C) we generate the branch (A,INSIDE,C) and repeat this process recursively until all the implied branches

are generated. As for relations, LOF, ABOVE, and INF, although they are transitive we do not generate the implied branches unless they are explicitly asked for.

5.3 Models and Graph Structure

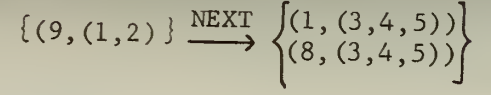
In defining our graph structure, which contains all the knowledge necessary for recognition, we start out with the definition of the PGS (primitive graph set). Each primitive region is represented as an n-tuple, where each index represents the primitive class of that attribute. For example (3,4) represents a primitive region which belongs to the third primitive class in respect to its shape attribute and fourth primitive class in respect to its compactness attribute, and this region is equivalent to all the regions in the real word, whose shape attribute is 3 and their compactness attribute has a value $8 \leq L < 10$. Since each primitive region has a set of attributes, and for each attribute value it belongs to a primitive class, we can have as many as 30 x 7 = 21 different primitive regions in our universe.

Each primitive node in a model graph represents a set of primitive regions which are acceptable as the primitive region represented by that node. Some primitive graphs (graphs which contain only primitive nodes) represent meaningful objects, while others are merely parts of higher level objects. Primitive graphs which do not represent any part of any higher level object belong to the FGS (final graph set). Using the non-final primitive graphs and primitive regions we construct the graphs which represent additional objects in our universe. This process is repeated until the graphical model of all the objects in the universe are constructed. In Table 5.2 we have shown the objects and their corresponding graphical models. In this table each super node is represented by its associated graph, and each

Table 5.2. Graphical models of the objects

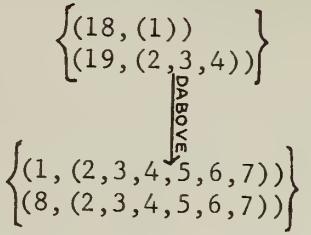
Model ID	Graph
M1	{(13,(2,3))} NEXT {(18,(1))} CARROT
M2	{(19,(1)), (21,(1))} DABOVE {(8,(1,2))} CONE
M3	{(3,(2,3,4)), (28,(2,3,4)), (12,(2,3,4))} NEXT {(1,(1)), (11,(1))} NEXT {(13,(2,3)), (1,(2,3))} KNIFE
M4	{(4,(1))} NEXT {(13,(2,3))} SPOON
M5	{(4,(1,2))} NEXT {(5,(1))} NEXT {(6,(1,2))} CUP
M6	{(2,(1,2)), (4,(1,2))} DABOVE, DABOVE, DABOVE, DABOVE {(1,(2,3,4,5,6)), (8,(2,3,4,5,6))} {*} {*} {*} TABLE
M7	{M6} DABOVE {(1,(2,3))} DESK
M8	{(1,(1,2)), (2,(1,2))} DABOVE {M6} CHAIR
M9	{(11,(1))} NEXT {(13,(3,4,5,6))} BAT
M10	{(6,(3,4,5,6,7))} NEXT {(1,(1))} BAG

M11



HAMMER

M12

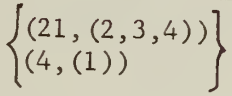


TREE

M13

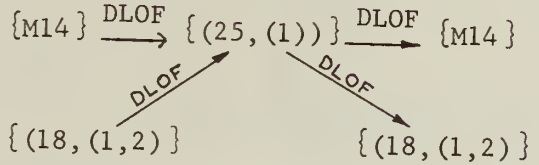
CLOUD

M14



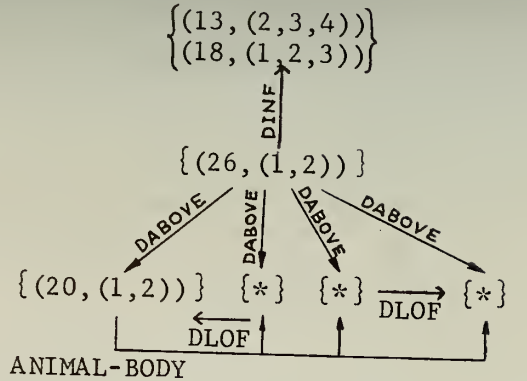
EYE

M15



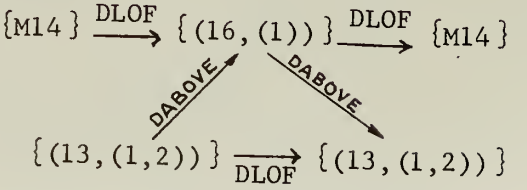
DOG-HEAD

M16



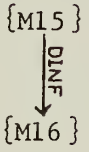
ANIMAL-BODY

M17



RABBIT-HEAD

M18



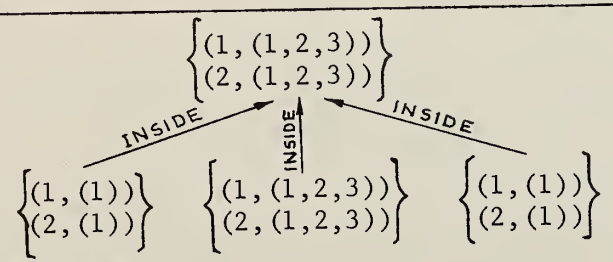
DOG

M19

RABBIT

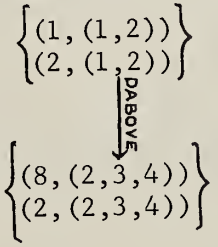
Table 5.2. Graphical models of the objects (continued)

M20



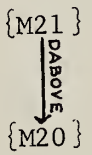
BUILDING

M21

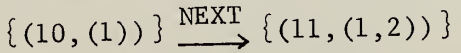


ROOF

M22

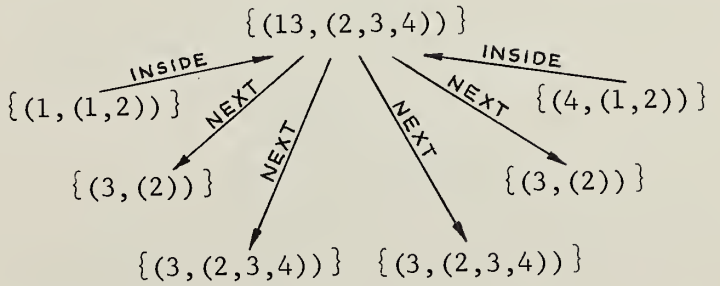


M23



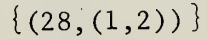
HAT

M24



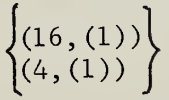
PLANE

M25



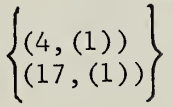
MOUNTAIN

M26



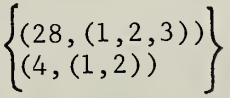
BIRD-SKULL

M27



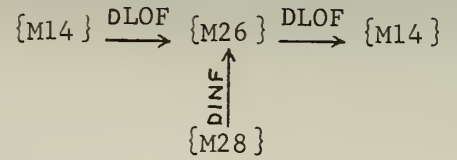
BIRD-B

M28



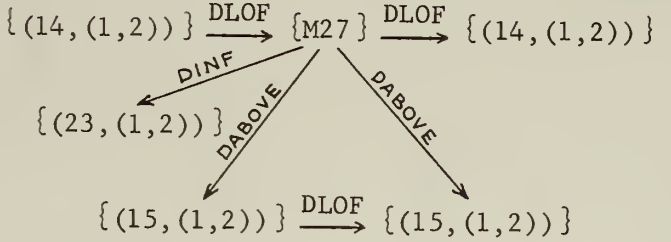
BIRD-BEAK

M29



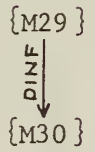
BIRD-HEAD

M30



BIRD-BODY

M31

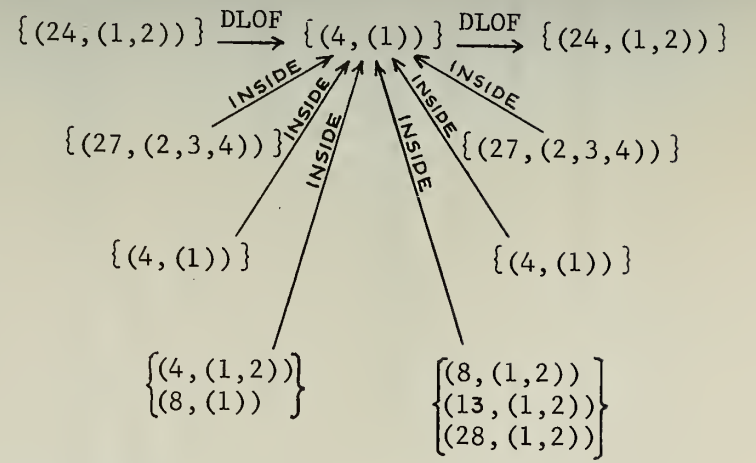


BIRD

M32

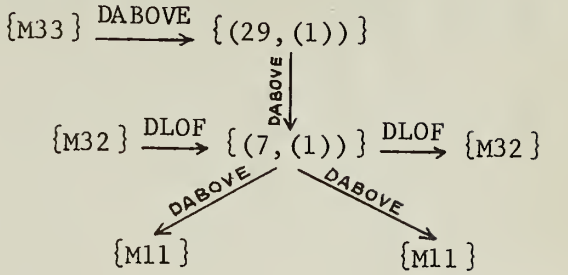
FOLK

M33



MAN-HEAD

M34



primitive node with a set of alternative primitive regions which it represents. For each alternative the first number inside the parentheses represents the shape primitive class, and the set of numbers inside the second parentheses represent the acceptable compactness primitive classes. In this representation a primitive node can be associated with several different regional views of the same part of the object.

Now that we have defined the graphs in our universe, we can define the entry points associated with the primitive classes as follows:

Entry points associated with compactness primitive classes:

1: M1,M2,M3,M4,M5,M6,M8,M9,M10,M12,M13,M14,M15,M16,M17,M20,M21,M23,M24,M25, M26,M27,M28,M30,M32,M33,M44

2: M1,M2,M3,M5,M7,M8,M11,M12,M13,M14,M16,M17,M20,M21,M23,M24,M25,M27,M28, M30,M33

3: M3,M4,M6,M9,M10,M11,M12,M13,M14,M16,M21,M24,M28,M32,M33

4: M3,M6,M9,M10,M11,M12,M13,M14,M16,M21,M24,M32,M33

5: M6,M9,M10,M11,M12,M13,M24,M32,M33

6: M6,M9,M10,M11,M12,M13

7: M10,M12,M13 .

Entry points associated with shape primitive classes:

1: M6,M7,M8,M11,M12,M20,M21,M24,M32

2: M6,M8,M20,M21

3: M3,M24

4: M4,M5,M6,M14,M24,M26,M27,M28,M33

5: M5

6: M5,M10

7: M34

8: M2,M6,M11,M12,M21,M32,M33

9: M11

10: M23

11: M3,M9,M23

12: M3

13: M1,M3,M4,M9,M17,M17,M33

14: M30

15: M30

16: M26

17: M27

18: M1,M12,M15,M16

19: M2,M12

20: M16

21: M2,M14

22: M13

23: M30,M32

24: M33

25: M15

26: M16

27: M33

28: M3,M25,M28,M33

29: M34 .

As we observe, the shape attribute provides us with rich information to pinpoint the set of entry points in our parsing graph. With some minor heuristics we are able to improve on this even further. For example if two structurally tied primitive nodes have shape attributes 1 and 18 respectively, their implied entry points will be

1: M6,M7,M8,M11,M12,M20,M21,M24,M32

18: M1,M12,M15,M16 , and

by intersecting these to sets;

1 and 18: M12 .

In this case we find that rule M12 should be tried first.

Fig. 5.1 shows the parsing graph of our graph structure; the best way to find out how the system works is to go through an example carefully. We do this in the next section.

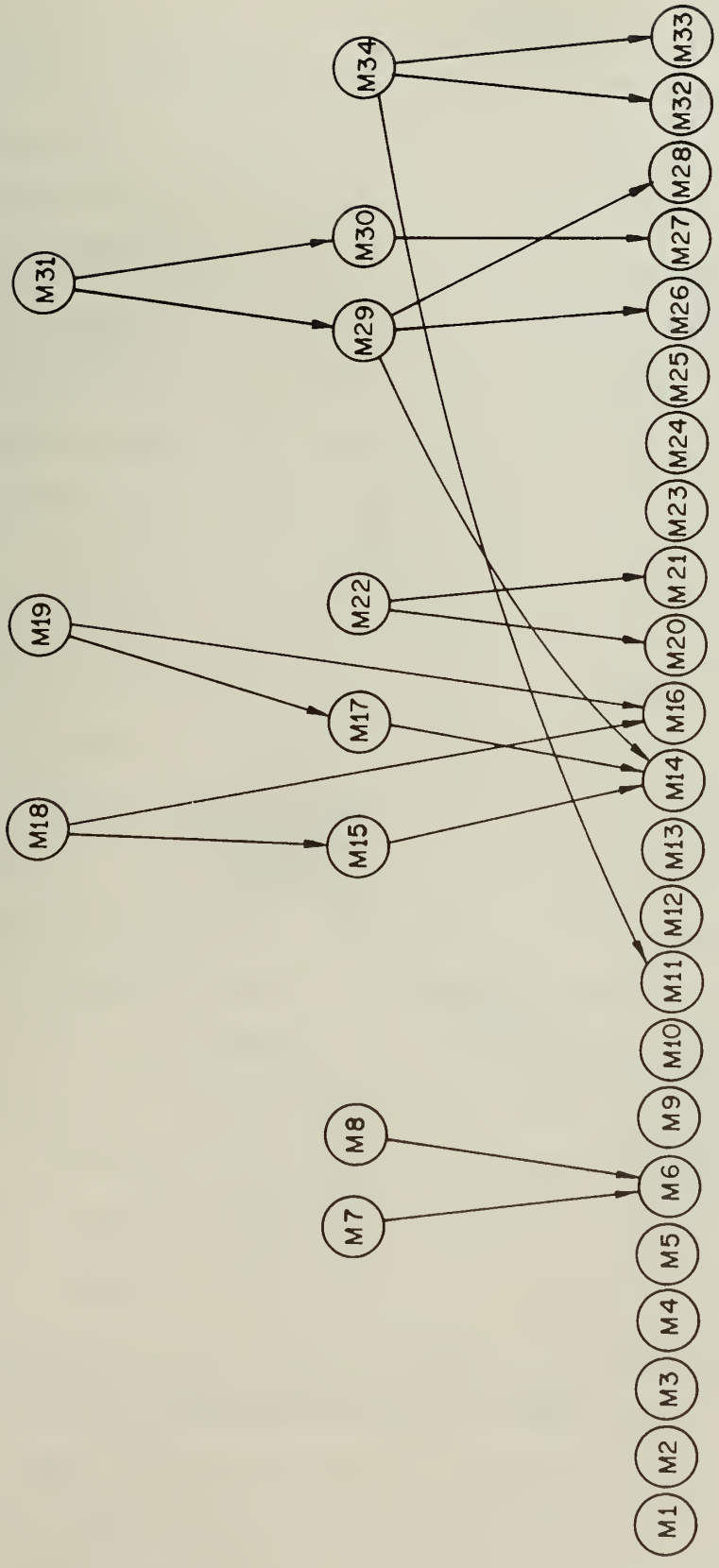


Fig. 5.1. Parsing graph for this application.

5.4 Careful Analysis of an Example

First to give a general view of the system's performance, we have shown the computer results of analyzing the scene in Fig. 5.2. The graphical representation of this scene is shown in Fig. 5.3. The computer processing time for this example was 21.40 seconds. This includes the preprocessing time for the "house", and rotation of the bird model, which will be discussed in later sections. The only heuristic used is the object associations, meaning that every time we discover an object in the scene we give higher priority to the objects which can jointly occur with this object in a natural scene. No backtracking was necessary in this analysis.

Since this scene is rather complex for detailed analysis, we have chosen one portion of it as shown in Fig. 5.4. The analysis time for this portion was 7.28 seconds, while the same portion required 8.45 seconds in a more complex environment of Fig. 5.2. This is mainly due to the linear list processing technique which requires more processing time in a larger scene.

Fig. 5.5(a) shows the scene graph after relation preprocessing, which replaces the inverse relations and creates branches implied by transitivity for relation "INSIDE". First AP (attention point) pointed out by the user was region 29 of Fig. 5.4 (or node "ND29" of Fig. 5.5(a)). The shape attribute of this region has the value 23, which implies rules M30 and M32. The value of compactness attribute can not reduce this set any further. Since the front view of bird's body (M30) did not match with a subgraph in the scene (starting at "ND29"), M30 was rotated 90° to the left, and matching was tried again. This time domain A1 of the scene graph matches with this rotated M30. Domain A1 was parsed to a super node "M30", shown in Fig. 5.5(b). Now rule M30 in the parsing graph implies rule M31. The matched domain with this rule is A3, and in the course of search for this domain we

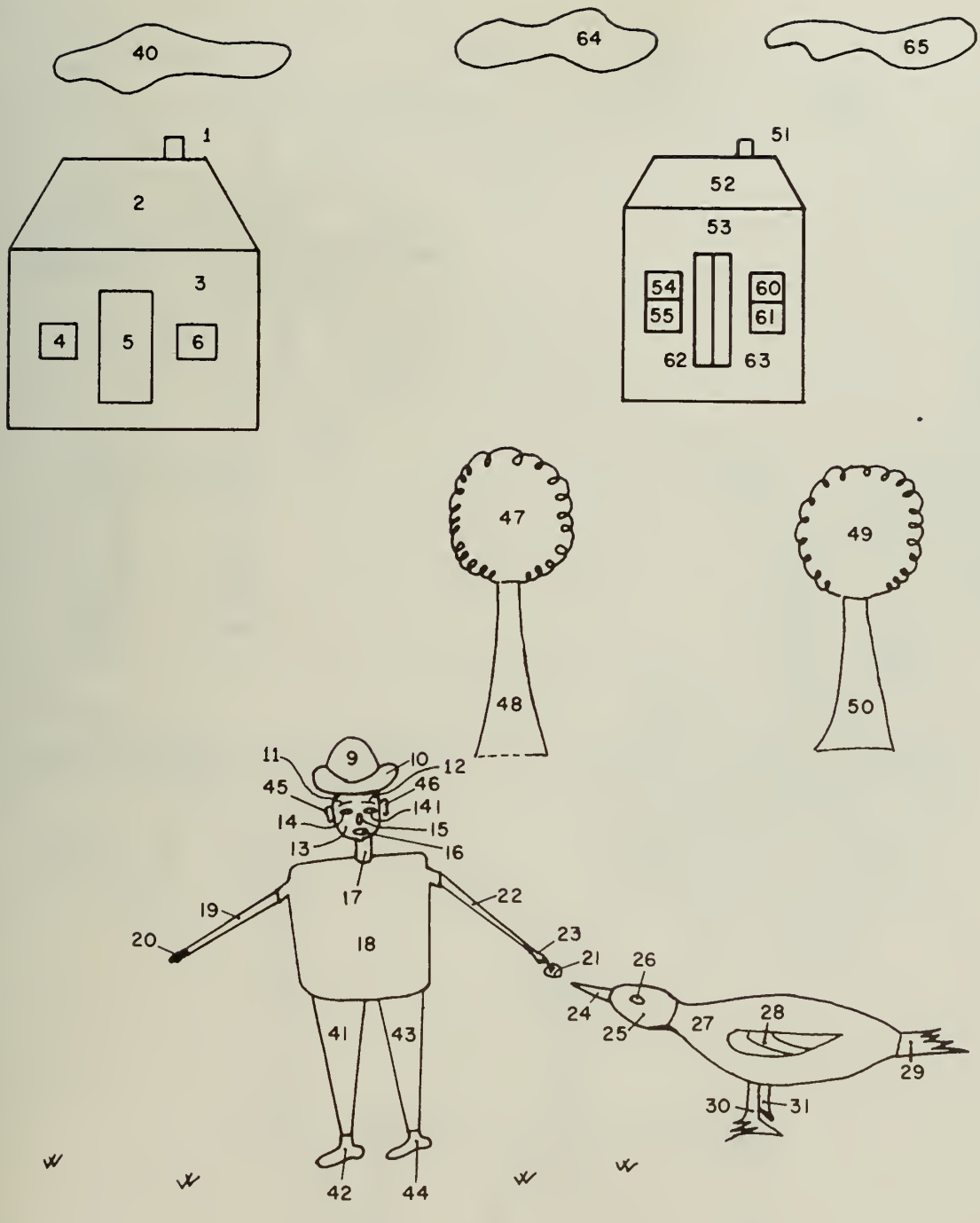


Fig. 5.2. A scene example. Numbers are used as node identifiers in the scene graph.

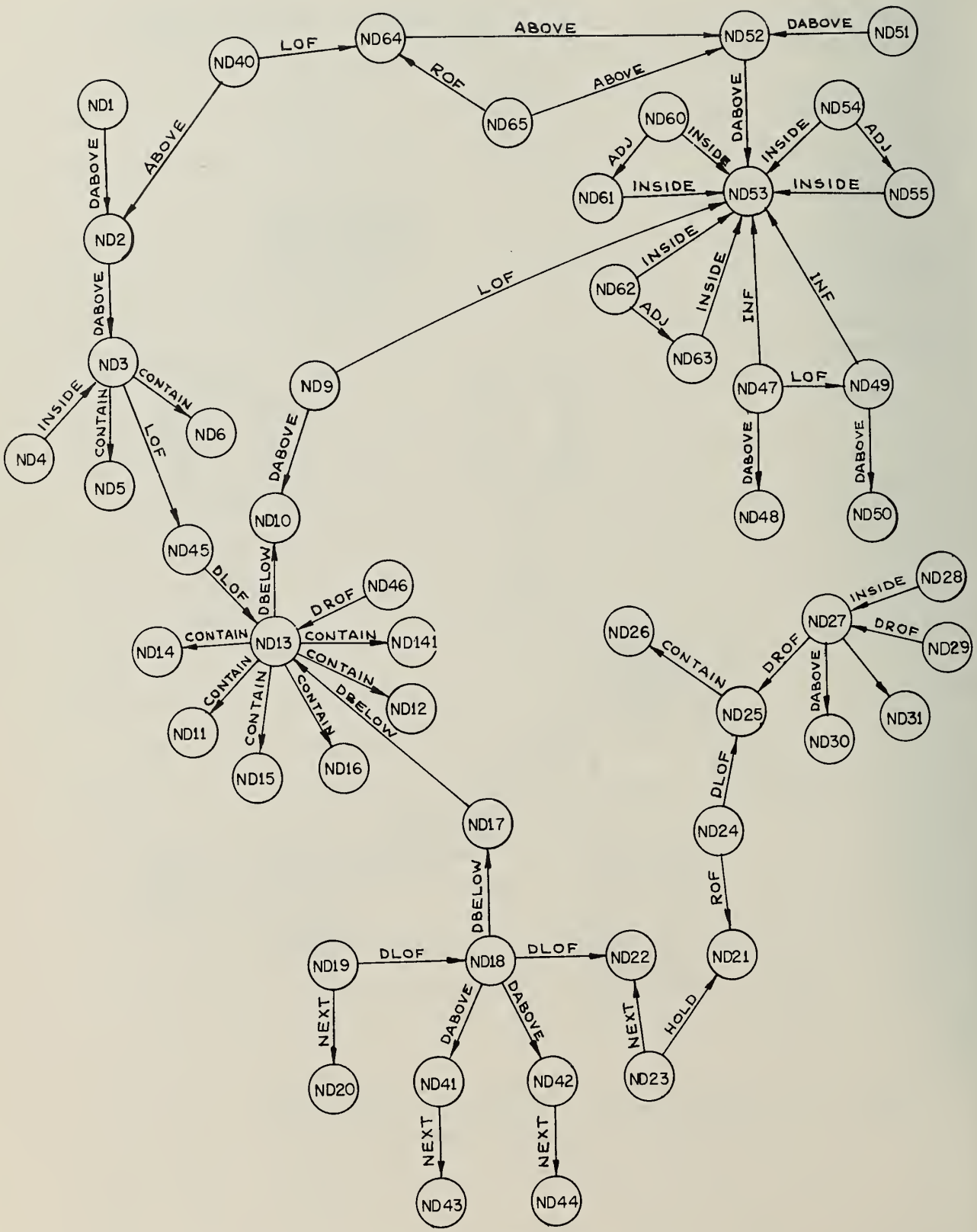


Fig. 5.3. Graphical representation of Fig. 5.2.

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. BIRD
2. HOUSE
3. MAN
4. CLOUD
5. HAT
6. HOUSE
7. CLOUD
8. CLOUD
9. TREE
10. TREE

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *TREE**.

REGION *ND49** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *ND47** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *ND65** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *ND64** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *HOUSE**.

OBJECT *BUILDING**.

REGION *ND53** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *HAT**.

REGION *ND10** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *ND40** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *MAN**.

OBJECT *HAND**.

REGION *ND19** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *HOUSE**.

OBJECT *ROOF**.

REGION *ND1** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

REGION *ND21** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_R**.

REGION *ND29** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

THE PRIMITIVE REGION *ND21** IS LEFT WITHOUT ANY INTERPRETATION.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

THE PRIMITIVE REGION *ND31** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *ND31** IS MOST LIKELY A HIDDEN PART OF THE *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

A *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** IS LOCATED AT THE LEFT OF THE PRIMITIVE REGION *ND21**.

THIS *BIRD** HAS THE STRUCTURAL TIE *DABOVE** TO THE PRIMITIVE REGION *ND31**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *HOUSE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *HOUSE** IS LOCATED AT THE LEFT OF THE *MAN**.

THIS *HOUSE** IS LOCATED BELOW THE *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *MAN** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THE RELATION *HOLD** BETWEEN THE *MAN** AND THE PRIMITIVE REGION *ND21** IS NOT KNOWN TO THE RECOGNIZER.

THIS *MAN** IS LOCATED AT THE RIGHT OF THE *HOUSE**.
THIS *MAN** IS LOCATED BELOW THE *HAT**.
THIS *MAN** IS LOCATED AT THE LEFT OF THE SECOND *HOUSE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED ABOVE THE *HOUSE**.
THIS *CLOUD** IS LOCATED AT THE LEFT OF THE SECOND *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *HAT** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *HAT** IS LOCATED ABOVE THE *MAN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *HOUSE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *HOUSE** IS LOCATED AT THE RIGHT OF THE *MAN**.
THIS *HOUSE** IS LOCATED BELOW THE SECOND *CLOUD**.
THIS *HOUSE** IS LOCATED BELOW THE THIRD *CLOUD**.
THIS *HOUSE** IS LOCATED IN BEHIND OF THE *TREE**.
THIS *HOUSE** IS LOCATED IN BEHIND OF THE SECOND *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED AT THE RIGHT OF THE *CLOUD**.
THIS *CLOUD** IS LOCATED ABOVE THE SECOND *HOUSE**.
THIS *CLOUD** IS LOCATED AT THE LEFT OF THE THIRD *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED ABOVE THE SECOND *HOUSE**.
THIS *CLOUD** IS LOCATED AT THE RIGHT OF THE SECOND *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *TREE** IS LOCATED IN FRONT OF THE SECOND *HOUSE**.
THIS *TREE** IS LOCATED AT THE LEFT OF THE SECOND *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *TREE** IS LOCATED IN FRONT OF THE SECOND *HOUSE**.
THIS *TREE** IS LOCATED AT THE RIGHT OF THE *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

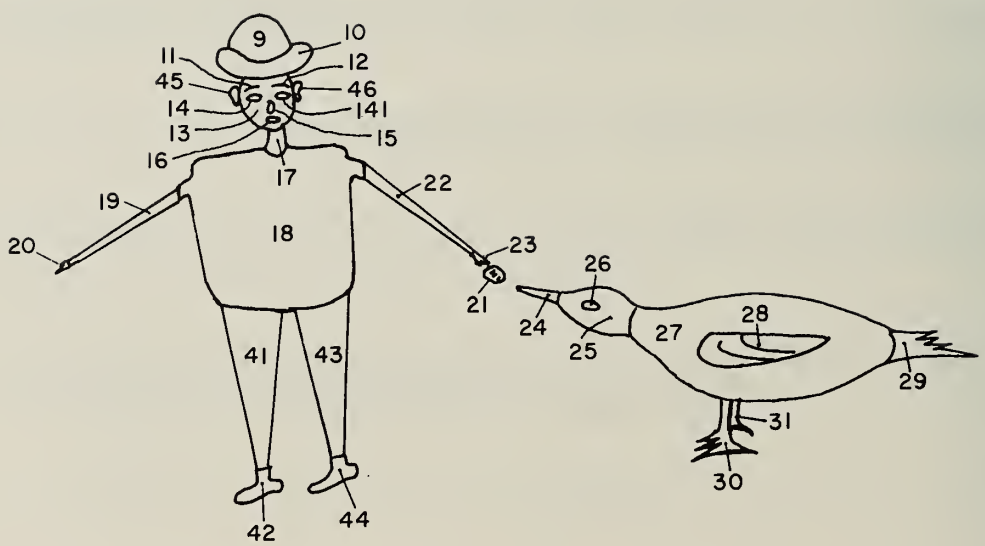
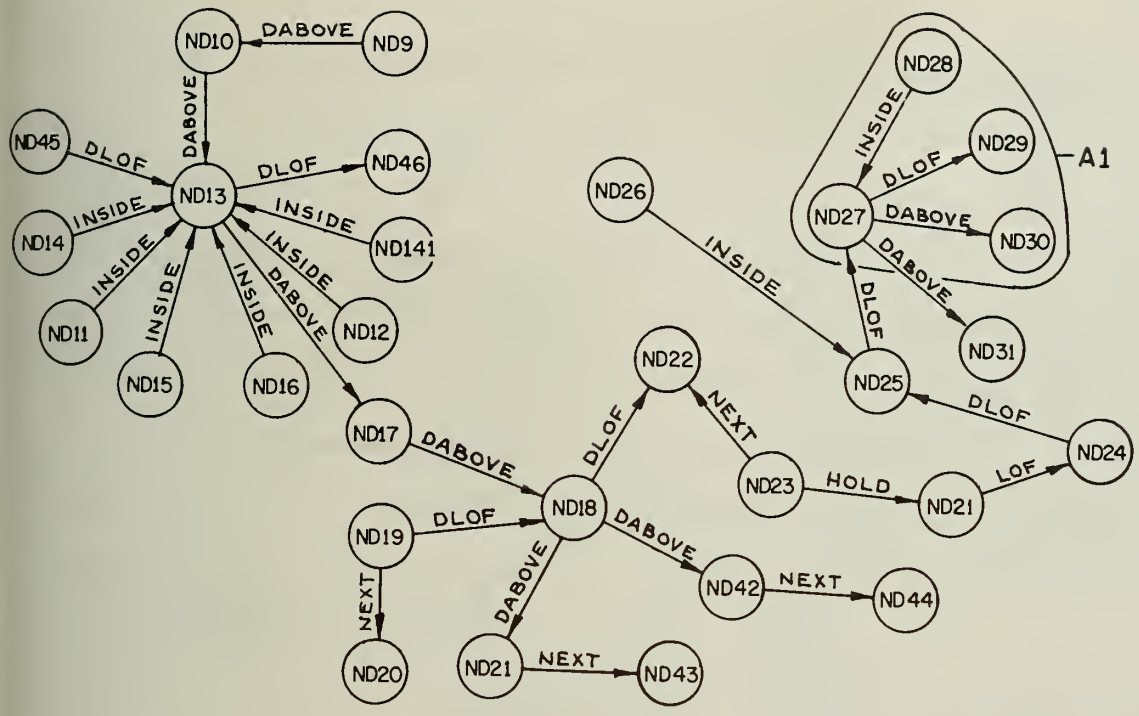
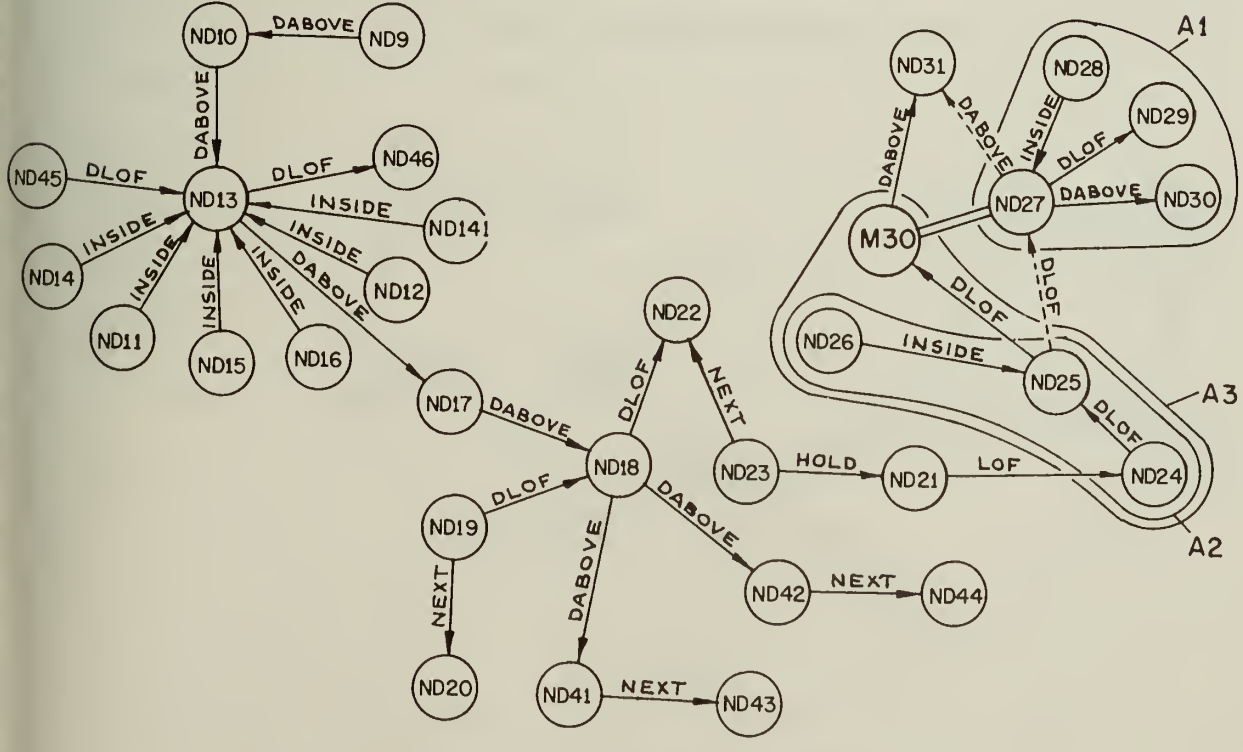


Fig. 5.4. A subpicture of the picture in Fig. 5.2.

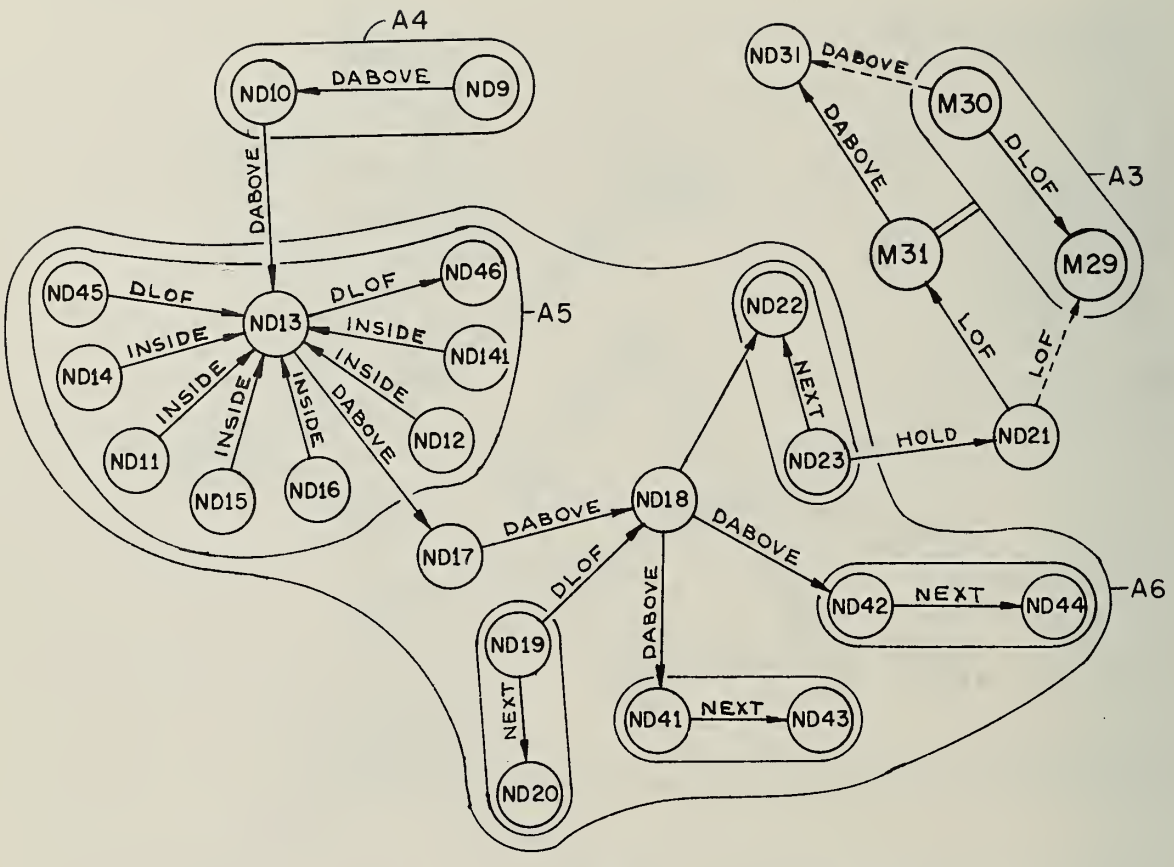


(a) graphical representation after relation preprocessing

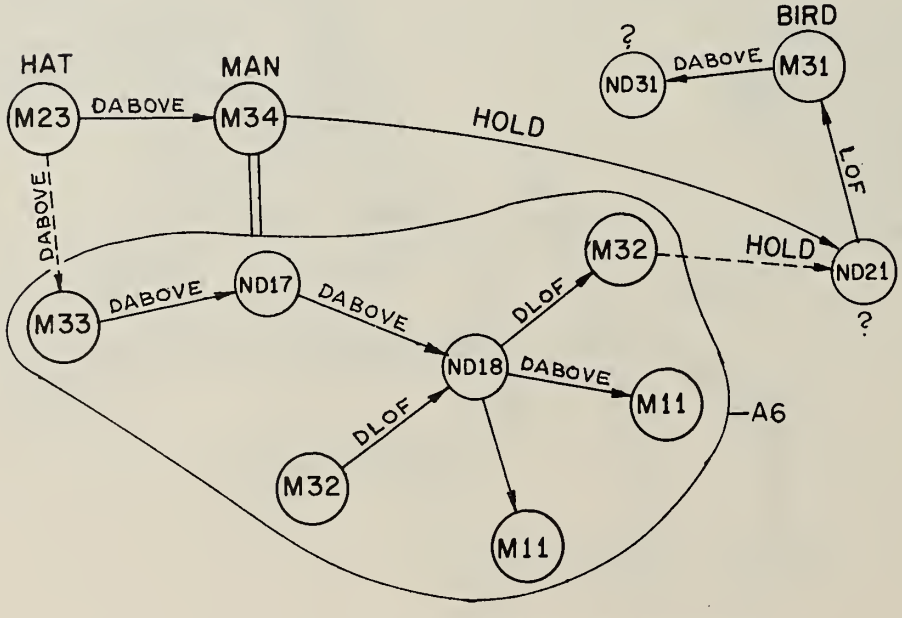


(b) scene graph after bird's body was parsed (M30)

Fig. 5.5. Part A. Parsing process of the scene in Fig. 5.4.



(c) scene graph after bird was parsed (M31)



(d) final processed scene graph

Fig. 5.5. Part B. Continued parsing of the scene in Fig. 5.4.

had to look for the subdomain A2 which corresponds to bird's head (M29). The domain of these parsed super nodes is eliminated from Fig. 5.5(c) for clarity. Since "M31" corresponds to a final graph ($\in$ FGS), search for this object is complete. The next attention point picked up by the recognizer was region 9 (node "ND9"), which is an excellent choice since its shape attribute has value 10, which implies one rule M23 (the hat model). Domain A4 in Fig. 5.5(c) corresponds to M23 which is parsed as such. Next we observe the relation "DABOVE" between this hat and region 13 (node "ND13"), which the value of its shape attribute (4) implies the following set of rules: M4, M5, M6, M14, M24, M26, M27, M28, M33.

But since the system is knowledgeable of object associations, it will find out that the man is associated with hat (semantically). So the following rules M33, M34, M11, M32 are given higher priority in the list of possible entry points from AP (in this case "ND13"). From these rules only M33 occurs in the above list, which has to be tried first. Indeed, this has been the case and domain A5 is found, which corresponds to M33 (man's head). Man's head in the parsing graph implies a man (M34). Domain A6 corresponds to this rule, besides A5 which has already been parsed. The algorithm had to look for 4 subdomains which correspond to two hands (M32) and two feet (M11). Domain A6 is parsed to a super node "M34". The recognizer also tried to parse nodes ND31 and ND21, but was unable to do so. ND31 had a structural tie to the BIRD, so the recognizer assumed that this must be a partially hidden part of the rotated bird. The final processed scene graph is shown in Fig. 5.5(d) which has 5 active nodes. The message routine will report this final graph. This report is reproduced in the following pages.

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. BIRD
2. HAT
3. MAN

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *MAN**.

OBJECT *FACE**.

REGION *ND13** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *HAT**.

REGION *ND9** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

REGION *ND21** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_B**.

REGION *ND29** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

THE PRIMITIVE REGION *ND21** IS LEFT WITHOUT ANY INTERPRETATION.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

THE PRIMITIVE REGION *ND31** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *ND31** IS MOST LIKELY A HIDDEN PART OF THE *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

A *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** IS LOCATED AT THE LEFT OF THE PRIMITIVE REGION *ND21**

THIS *BIRD** HAS THE STRUCTURAL TIE *ABOVE** TO THE PRIMITIVE REGION *ND31**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *HAT** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *HAT** IS LOCATED ABOVE THE *MAN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *MAN** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THE RELATION *HOLD** BETWEEN THE *MAN** AND THE PRIMITIVE REGION *ND21** IS NOT KNOWN TO THE RECOGNIZER.

THIS *MAN** IS LOCATED BELOW THE *HAT**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

5.5 Other Features of the Recognizer

The recognizer has several excellent features which enables it to recognize incomplete objects, different views of the objects and varieties of the same object. It is also able to cope with the preprocessors which divide scenes into three-dimensional bodies. We will discuss these in the following subsections.

5.5.1 Recognition of incomplete objects

In chapter 4 we discussed the best match feature of our recognizer. Here we give an example of scene, Fig. 5.6, where one of the legs of table and chairs and a dog's foot is hidden (or unrecognizably partially hidden) from the view. For chairs and the table the search has been very expensive in time, since all the entry points implied by the AP's had to be tried before M6 was picked as the best match. As for the dog, the head was recognized first, and the body's partial match with FOM (figure of merit) 5/6, which is greater than AFOM (acceptable figure of merit) (.75), was recognized immediately. The recognizer also realized that region 30 has a structural tie to this partially matched body and assumed that this must be a partially hidden leg. The report of this experiment is reproduced in the following pages.

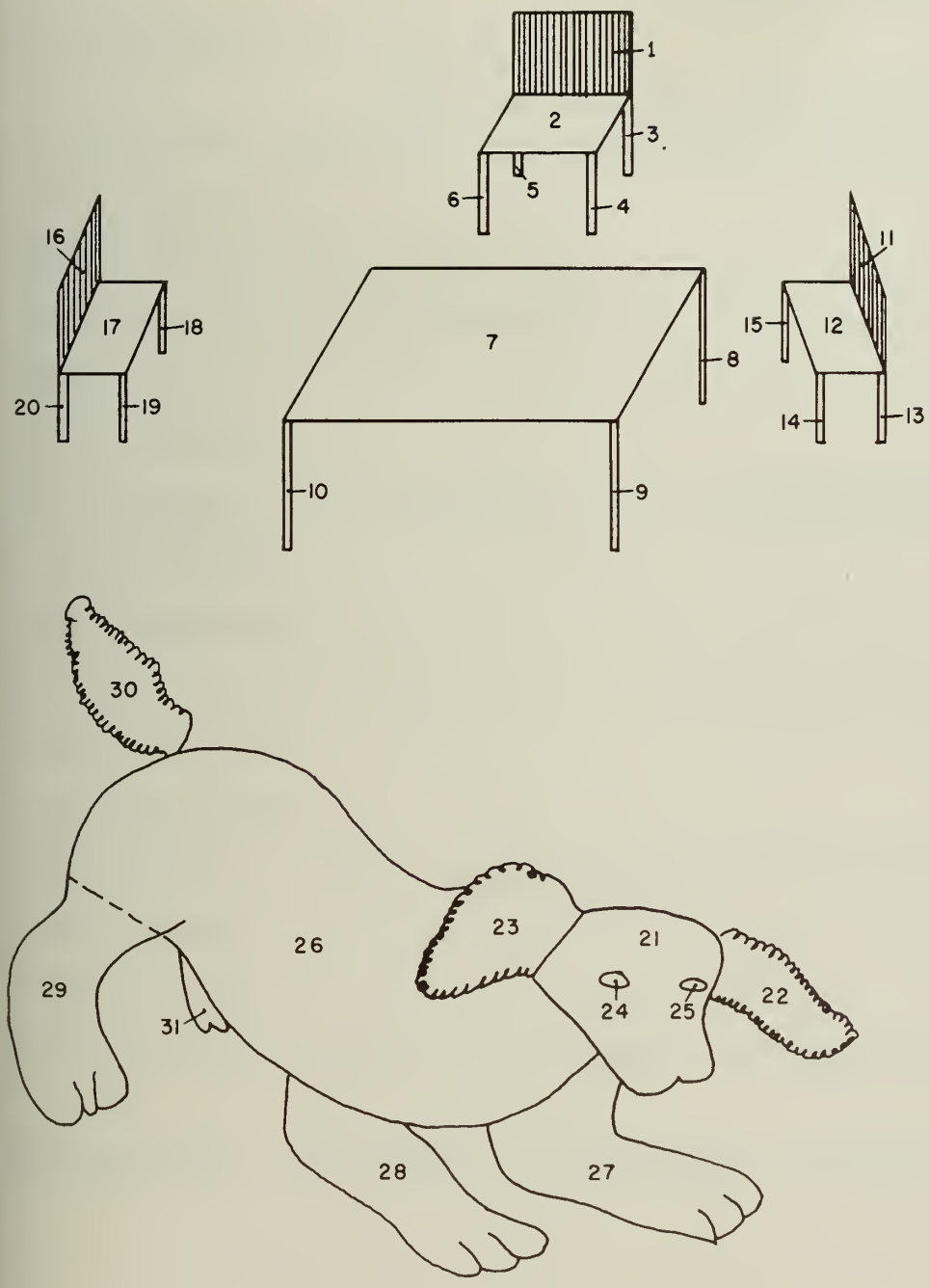


Fig. 5.6. A scene where objects have (partially hidden unrecognizable parts.

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. DOG
2. TABLE
3. CHAIR
4. CHAIR
5. CHAIR

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *CHAIR**.

OBJECT *MODEL6**.

REGION *ND18** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CHAIR**.

OBJECT *MODEL6**.

REGION *ND4** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CHAIR**.

OBJECT *MODEL6**.

REGION *ND15** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TABLE**.

OBJECT *MODEL6**.

REGION *ND10** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *DOG**.

OBJECT *DOG_H**.

REGION *ND21** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

THE PRIMITIVE REGION *ND31** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *ND31** IS MOST LIKELY A HIDDEN PART OF THE *DOG**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

A *DOG** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *DOG** HAS THE STRUCTURAL TIE *DABOVE** TO THE PRIMITIVE REGION *ND31**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

THIS *DOG** IS LOCATED IN FRONT OF THE *TABLE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *TABLE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *TABLE** IS LOCATED IN BEHIND OF THE *DOG**.

THIS *TABLE** IS LOCATED AT THE LEFT OF THE *CHAIR**.

THIS *TABLE** IS LOCATED IN FRONT OF THE SECOND *CHAIR**.

THIS *TABLE** IS LOCATED AT THE RIGHT OF THE THIRD *CHAIR**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *CHAIR** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *CHAIR** IS LOCATED AT THE RIGHT OF THE *TABLE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *CHAIR** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CHAIR** IS LOCATED IN BEHIND OF THE *TABLE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *CHAIR** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CHAIR** IS LOCATED AT THE LEFT OF THE *TABLE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

5.5.2 Scenes with varieties of the same object

In describing the graphical models of the objects in the universe we allowed that more than one class of primitive region be associated with a primitive node. In testing the semantical equivalence of primitive nodes, all these alternatives are tried and a single equivalence establishes the match. In Fig. 5.7 we have shown a scene with different varieties of the object "KNIFE". The results of analyzing this scene are shown in the following pages. Of course it would be trivial to add the actual shape information about each part of the objects to this report.

We can also easily extend this idea to super nodes and have more than one object (graphical rule) associated with each super node.

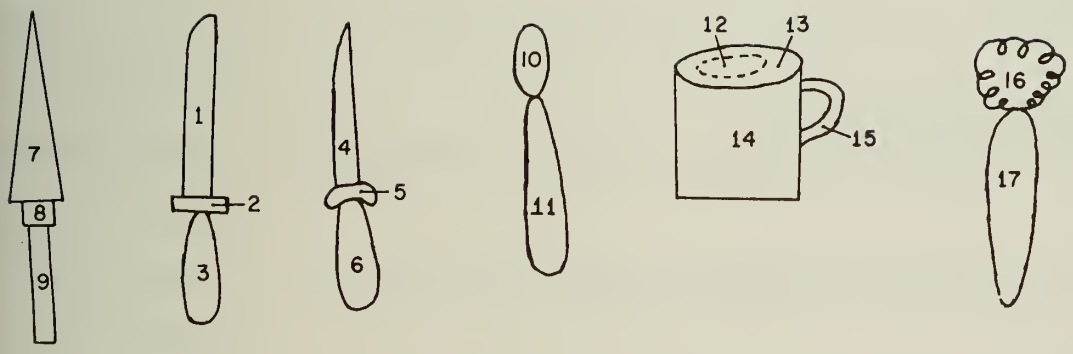


Fig. 5.7. A scene with varieties of the same object.

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. KNIFE
2. KNIFE
3. KNIFE
4. SPOON
5. CUP
6. CARROT

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *CARROT**.

REGION *N16** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CUP**.

REGION *N14** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *SPOON**.

REGION *N11** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *KNIFE**.

REGION *N4** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *KNIFE**.

REGION *N9** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *KNIFE**.

REGION *N1** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

THE PRIMITIVE REGION *N12** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *N12** IS MOST LIKELY A HIDDEN PART OF THE *CUP**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

A *KNIFE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *KNIFE** IS LOCATED AT THE RIGHT OF THE SECOND *KNIFE**.

THIS *KNIFE** IS LOCATED AT THE LEFT OF THE THIRD *KNIFE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *KNIFE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *KNIFE** IS LOCATED AT THE LEFT OF THE *KNIFE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *KNIFE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *KNIFE** IS LOCATED AT THE RIGHT OF THE *KNIFE**.

THIS *KNIFE** IS LOCATED AT THE LEFT OF THE *SPOON**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *SPOON** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *SPOON** IS LOCATED AT THE RIGHT OF THE THIRD *KNIFE**.

THIS *SPOON** IS LOCATED AT THE RIGHT OF THE *CUP**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *CUP** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *CUP** HAS THE STRUCTURAL TIE *INSIDE** TO THE PRIMITIVE REGION *N12**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

THIS *CUP** IS LOCATED AT THE LEFT OF THE *SPOON**.

THIS *CUP** IS LOCATED AT THE LEFT OF THE *CARROT**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *CARROT** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *CARROT** IS LOCATED AT THE RIGHT OF THE *CUP**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

5.5.3 Preprocessing of the scene graph

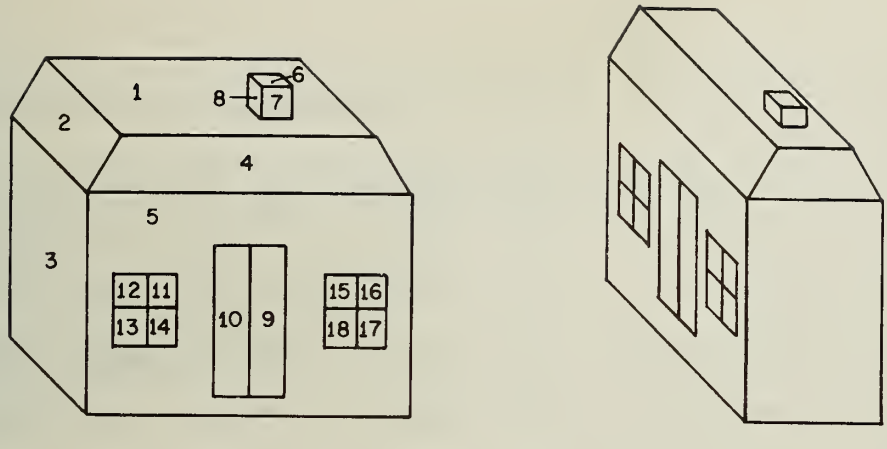
The nodes in graphical rules correspond to the meaningful parts of the objects. These nodes normally represent the regional views of the parts. In actual 3-dimensional scenes, however, there are cases where more than one regional view of the parts are visible. So, we need a class of rules, which operate on the actual scenes and conform them to our one-node-representation of parts. This is accomplished through MERGING of the nodes in the scene graph.

Guzman [19] has been pioneer in discovering rules to divide scenes into 3-dimensional bodies. Fig. 5.8(a) shows two different views of the same "house" where several regions of each part are visible. Using simple vertex properties like "T-joint" and "Y-joint", we can find the collection of regions which correspond to each part. Then, we can merge these nodes into a node equivalent to the primitive node in the model graph. In the merging process the structural relations are embedded irredundantly, and the association of the newly created node is simply the union of associations of the merged nodes.

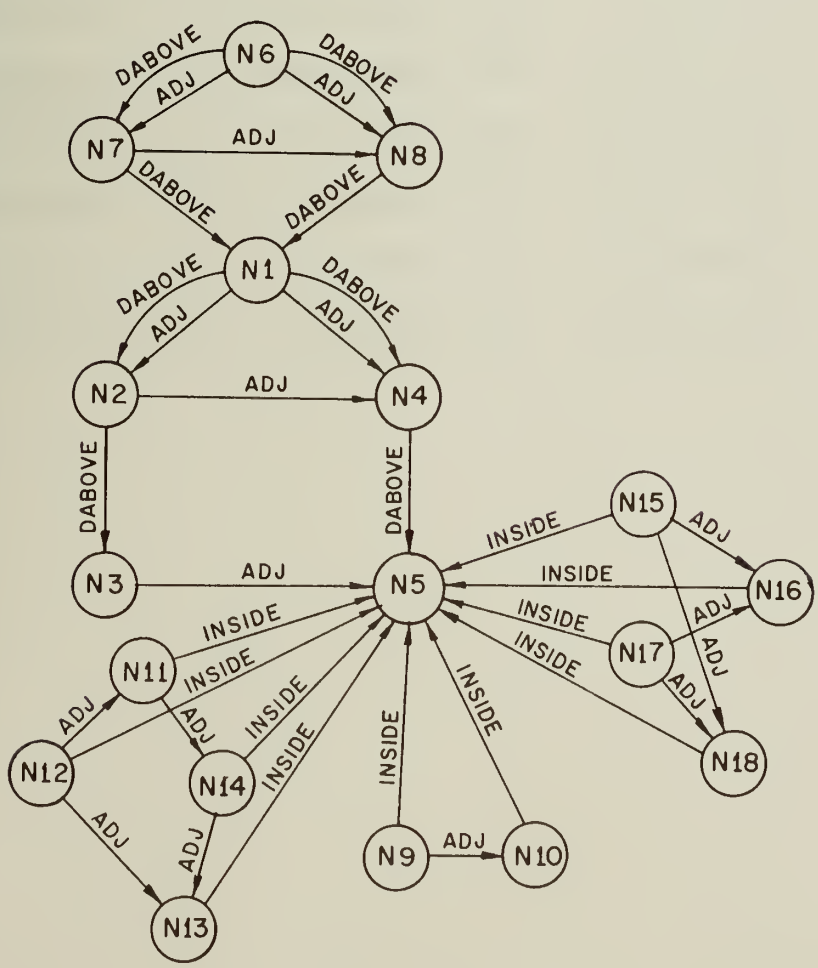
Since the parts of a house are symmetric, the graphical representation of both vies will be the same, Fig. 5.8(b).

The results of parsing this scene are reproduced in the following pages. The first page gives the merging operations which are performed before the recognizer is called.

In Appendix B we have represented another example of this type, where the regions of a "chair" have been divided into several subregions.



(a) picture



(b) input graph

Fig. 5.8. An example of a house with complex subparts.

NODES N2 AND N1 ARE MERGED INTO ONE NODE,NAMED GEN11

NODES N7 AND N6 ARE MERGED INTO ONE NODE,NAMED GEN12

NODES N5 AND N3 ARE MERGED INTO ONE NODE,NAMED GEN13

NODES N10 AND N9 ARE MERGED INTO ONE NODE,NAMED GEN14

NODES N12 AND ND11 ARE MERGED INTO ONE NODE,NAMED GEN15

NODES N14 AND N11 ARE MERGED INTO ONE NODE,NAMED GEN16

NODES N16 AND N15 ARE MERGED INTO ONE NODE,NAMED GEN17

NODES N18 AND N17 ARE MERGED INTO ONE NODE,NAMED GEN18

NODES N4 AND GEN11 ARE MERGED INTO ONE NODE,NAMED GEN19

NODES N8 AND GEN12 ARE MERGED INTO ONE NODE,NAMED GEN20

NODES N13 AND GEN15 ARE MERGED INTO ONE NODE,NAMED GEN21

NODES GEN18 AND GEN17 ARE MERGED INTO ONE NODE,NAMED GEN22

NODES GEN16 AND GEN21 ARE MERGED INTO ONE NODE,NAMED GEN23

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. HOUSE

END OF BRIEF DESCRIPTION.

EULL DESCRIPIION OE IHE SCENE.

IHE EOLLOWING SUCCESSEUL SIEPS WERE IAKEN IN PARSING IHE SCENE.

OBJECT *HOUSE**.

OBJECT *BUILDING**.

REGION *GEN13** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

IHIS CONCLUDES IHE REPORI OE IHE SUCCESSEUL AIIEMPIS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

A *HOUSE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

5.5.4 Recognition of the different views of an object

In chapter 4 we introduced a class of transformations T_{R90}, T_{R180}, and T_{R270} which operate on the graphical models of the objects and produce the models of different views of that object. In representing a part of an object as a primitive node in the graphical rule, we include all different regional views of that part tagged with angular information as acceptable primitive regions for that primitive node. And, besides the attributes discussed earlier we associate a "relative size" attribute with each node which gives the relative size of that part to the other parts of the object.

Now, production of different views of an object is merely the problem of applying the transformations T_1, T_2, or T_3 to the set of binary relations in the graph and deleting any nodes which at the new position are directly behind a node of greater relative size.

Fig. 5.9 shows four different views of a bird. In Fig. 5.10 we have shown the results of applying these transformations to the model of bird's body. Here we have introduced another transformation which merely deletes the hidden parts from the original model. When a part of the object is recognized we also will have the orientation information, and in further parsing we will use this information to transform the rules before any matching is attempted.

In the following pages we have reproduced the result of analyzing the scene of Fig. 5.9. For example in recognition of the bird's rear end, the body is first recognized with 180° orientation. In an attempt to find a match for the bird which consists of body and head, the rule is first transformed by T_{R180}, and the super node corresponding to the head is eliminated. So no further search for the head will be necessary.

It is also trivial to produce this rotational information in the report.

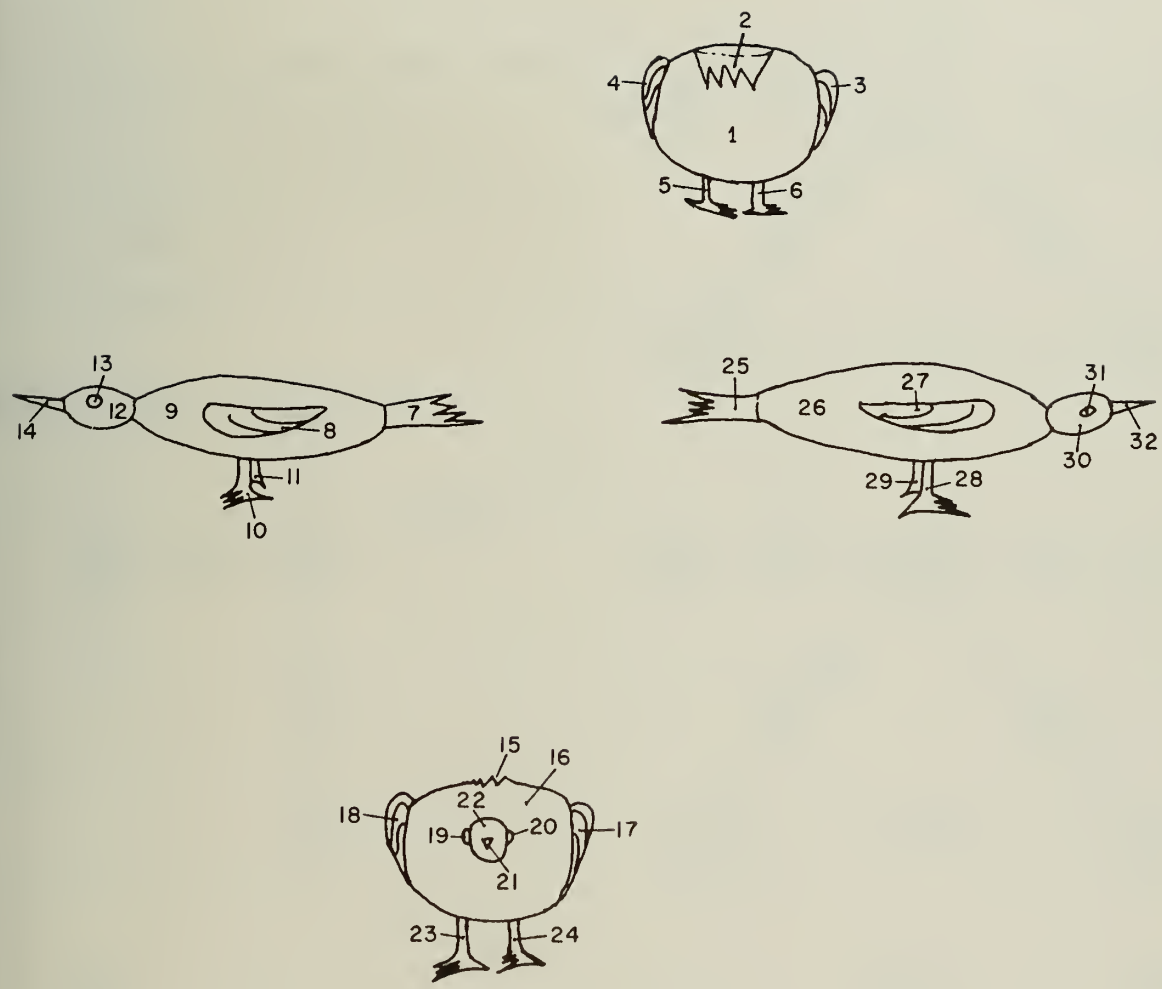


Fig. 5.9. Four different views of a bird.

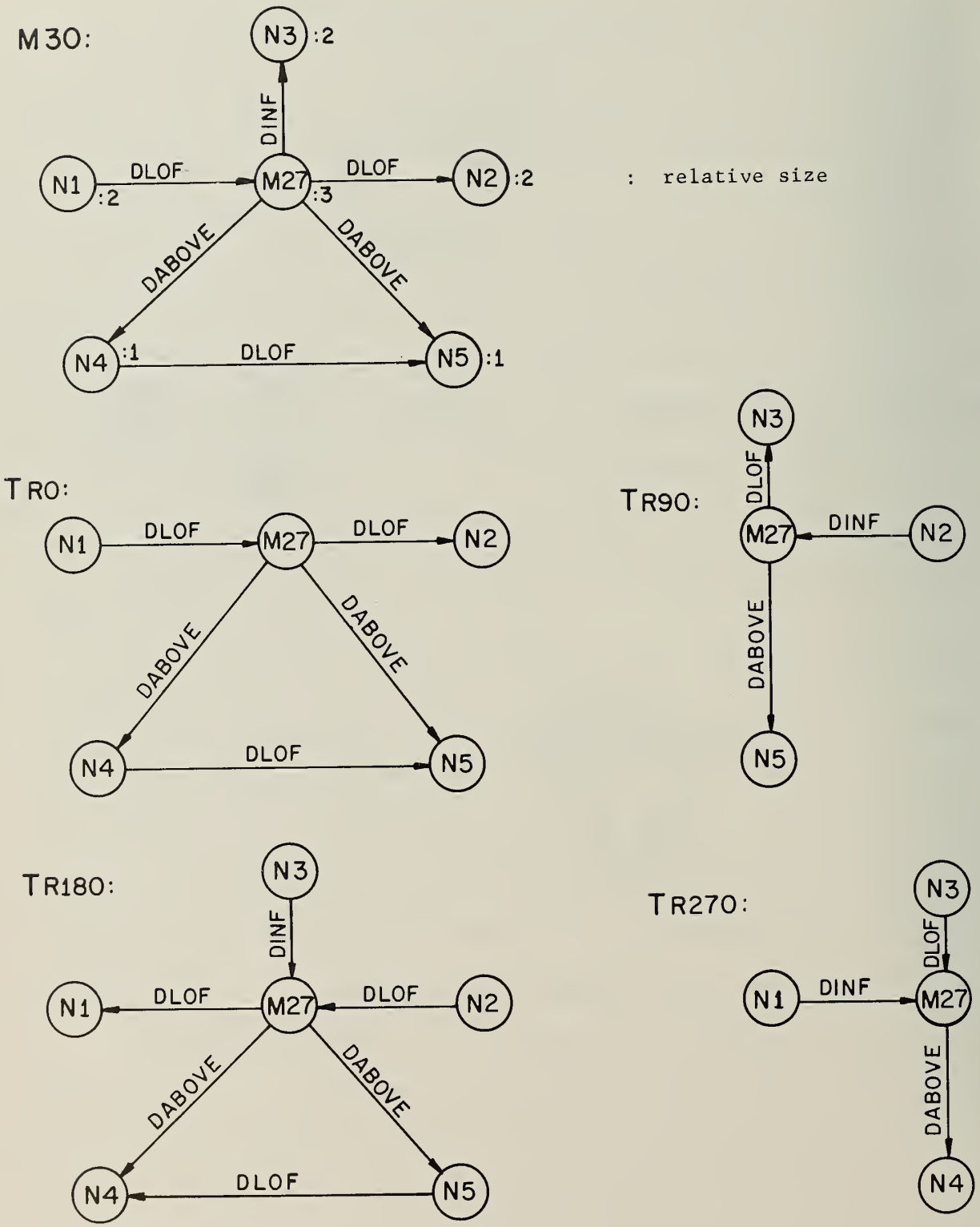


Fig. 5.10. Application of rotational transformations to rule M30 (bird's body).

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. BIRD
2. BIRD
3. BIRD
4. BIRD

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

REGION *N11** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_B**.

REGION *N25** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_B**.

REGION *N15** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_B**.

REGION *N7** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *BIRD**.

OBJECT *BIRD_B**.

REGION *N2** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

THE PRIMITIVE REGION *N11** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *N11** IS MOST LIKELY A HIDDEN PART OF THE SECOND *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

THE PRIMITIVE REGION *N29** IS LEFT WITHOUT ANY INTERPRETATION.

THE PRIMITIVE REGION *N29** IS MOST LIKELY A HIDDEN PART OF THE FOURTH *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS REGION.

A *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** IS LOCATED AT THE RIGHT OF THE SECOND *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** HAS THE STRUCTURAL TIE *DABOVE** TO THE PRIMITIVE REGION *N11**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

THIS *BIRD** IS LOCATED AT THE LEFT OF THE *BIRD**.

THIS *BIRD** IS LOCATED ABOVE THE THIRD *BIRD**.

THIS *BIRD** IS LOCATED AT THE LEFT OF THE FOURTH *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** IS LOCATED BELOW THE SECOND *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A FOURTH *BIRD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *BIRD** HAS THE STRUCTURAL TIE *DABOVE** TO THE PRIMITIVE REGION *N29**. THIS REGION COULD BE A HIDDEN PART OF THE OBJECT.

THIS *BIRD** IS LOCATED AT THE RIGHT OF THE SECOND *BIRD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

5.6 Observations

Because of a great many contextual factors involved in the recognition speed, we would need to experiment with many examples to produce any meaningful statistics.

The experiments have been carried out using 360/75 and our implementation of SOL. The graph generation and relation preprocessing are quite fast and in most cases took less than one second in execution time. Since our SOL execution time routines operate on Pℓ/1 pointers rather than offsets, we had to read the graph structure of our universe from the saved area for each experiment. This takes approximately 9 seconds.

The recognizer was written in SOL; its generated Pℓ/1 program is about 950 statements long. The subgraph matching routine was also written in SOL and its generated Pℓ/1 program is 600 statements long. There are a dozen other SOL programs whose algorithms have been discussed earlier, and they are very short and straightforward programs, provided that they are written in SOL.

In Table 5.3 we have shown the results of a few experiments, some of them mentioned earlier. Slow-core has been used in these experiments.

In Fig. 5.11 we have depicted the recognition time of the scenes via the number of nodes in the scene graph. Here we have excluded the examples which would need extra time to use special features of the recognizer. For example, experiment 7 would involve extensive rotations. With a fixed universe, this graph should be linear, but here it is slightly curved upward because the increased number of nodes will slow down the linear list searches.

In Fig. 5.12, we have depicted the average processing time for each object, against the average number of nodes per object of the scenes. Here again we have excluded the examples which have used the semantic association to reduce the recognition time or examples with extensive rotations. This curve is also approximately linear, because our domain search algorith discussed in chapter 4 is basically linear.

Experiment ID	# of nodes	# of objects	Go step time(s)	Recognition time(s)	Figure ID
1	5	1	10.67	1.69	
2	7	1	13.57	3.59	
3	17	6	12.94	4.08	Fig. 5.7
4*	27	9	22.20	13.32	
5	30	3	17.35	7.28	Fig. 5.4
6*	30	4	36.69	26.50	Fig. 5.6
7*	32	4	23.19	13.43	Fig. 5.9
8	50	9	35.07	21.40	Fig. 5.2
9	70	15	39.72	28.42	Appendix B

Table 5.3. Tabulated results of experiments.

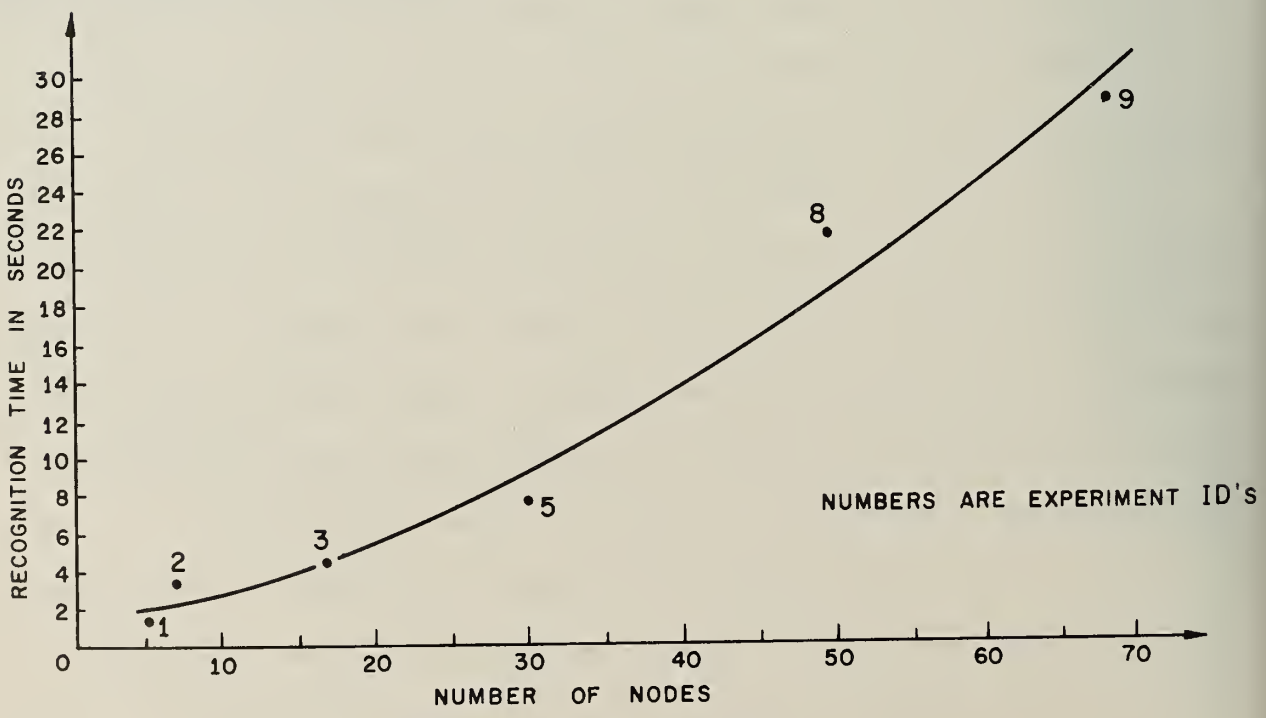


Fig. 5.11. Speed of recognition via # of nodes in the scene graph.

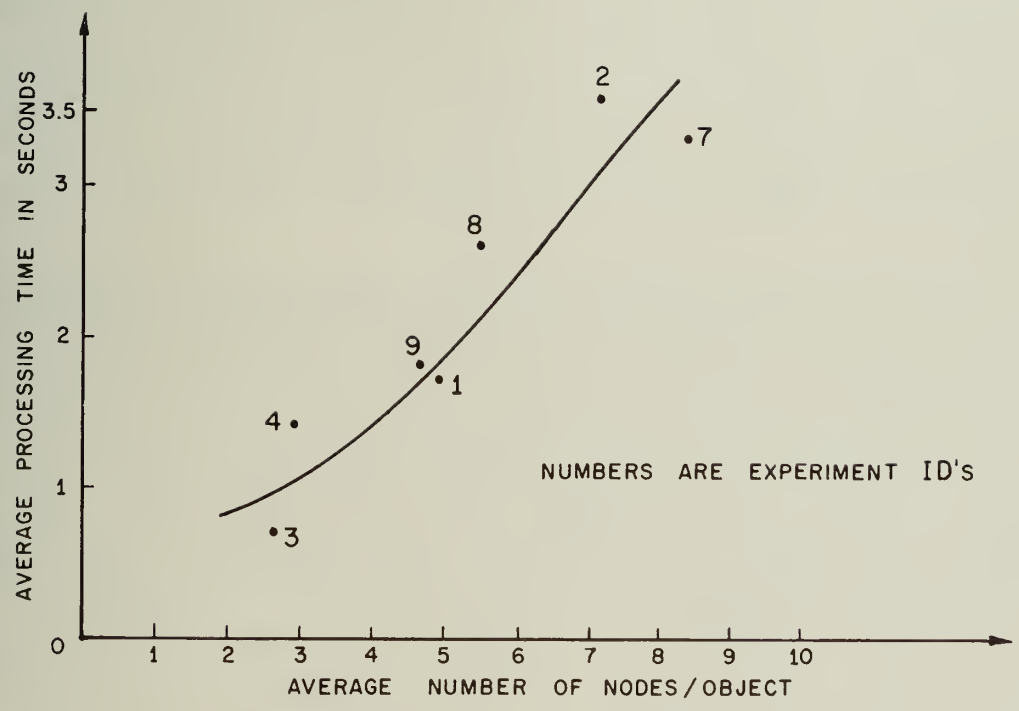


Fig. 5.12. Average processing time per object via # of nodes/object.

6. LEARNING

Another important factor in any intelligent system is the ability to learn. One aspect of artificial intelligence which has been more disturbing to outsiders than to insiders has been the apparently small degree of "learning" in the programs designed to solve the problems. These programs do not learn how to solve the problems; the methods are built-in. Of course, the matter is relative to one's goals: to make a machine with intelligence is not necessarily to make a machine that learns to be intelligent.

Many of the programs which people have not considered to be "learning" programs have an enormous "learning potential" just below the surface. Consider the qualitative effect upon the subsequent performance of Bobrow's STUDENT [23] of telling it that "distance equals speed times time"! That one experience alone enables it to handle a large new portion of high school algebra: the physical position-velocity-time problems. It is important not to get the habit, suggested by modern work in psychology, of concentrating only on the kinds of "learning" that appear as show-improvement-attendant-upon-sickeningly-often-repeated experience! Bobrow's program does not have any cautious statistical devices that have to be told something over and over again, so its learning is too brilliant to be called so.

Looking at it this way should clarify why we have come to feel that questions like "Why don't you put some learning into your program?" are much less sensible and straightforward than they may seem. In the early experiments in cybernetics the program's abstract knowledge was very small and most decisions were based upon the values of simple explicit parameters. When things are done that way one could always build in a crude sort of

adaptive behavior by using any of a number of correlation-like "reinforcement" schemes. There is no reason to suppose that anything like this is appropriate for "thinking". In thinking, the result of an intellectual experience is used, not simply to adjust a parameter but to construct a new way to represent something, or even to make a change in an administrative aspect of the problem-solving control system. Before it is profitable to attempt this, we need more experience with the systems that at least partially analyze their own problem-solving experiences. Then, we will probably discover the general underlying principles to these highly intelligent activities. One should not expect, however, to find problem-solving generality through the discovery of a single, magnificiently general problem-solving method. Even humans do not have unfailing success in new areas; the acquisition of a new problem-solving method is a major event in our own cultural evolution. Our approach here is to understand how the people can learn what they are told and to model these processes by programming the computers to achieve the same goal.

We define the learning ability of a program in a narrower sense as "performance improvement from one run to another". Our recognizer, as we stated before, functions in two different modes, namely learning and non-learning modes.

In the non-learning mode, the saved information is statistical in nature, and can affect only the ordering of the generated lists in the recognition process. This is a slow learning process through the accumulation of the analyzed results of the scene examples. These statistics can be classified in three different categories as follows:

a. There is a set of pointers (pointing to entry points) associated with each primitive class. By assigning a frequency attribute to each of

these pointers, we can order this set of entry points according to the values of these attributes. At each successful attempt the recognizer will increase the value of this attribute for all the pointers in the involved primitive classes pointing to this entry point. This in effect will change the ordering of the set of entry points implied by each primitive class, and cause the more frequently occurring entry points to be tried first. This will improve the recognition performance in a repetitious environment.

b. We can also have these frequency attributes associated with the branches in our parsing graph. At each successful attempt the value of this attribute for the branch which led us to this successful graphical rule will be increased. Now, if we order the set of successors according to the value of the frequency attribute of branch leading to each successor, we will be trying the rules in the sequence that more frequently occurring rules are tried first. This will also improve the recognition performance.

c. One of the useful heuristics in our recognition algorithm was the semantic association between the objects of the universe. For each object a set of pointers pointing to the objects (graphical rules), which occur commonly in natural scenes with this object, are associated with its graphical representation. We can have the frequency attributes associated with these pointers and increase the value of this attribute for those pointers which led us to objects actually occurred in conjunction with this object. This again can affect the order in which these objects will be tried, so that the system will adapt itself to the environment.

We see that even in non-learning mode the system is adaptive and increases its recognition for frequently occurring scenes. The amount of saved information in this mode is negligible compared to that in the learning mode.

In the learning mode our recognizer is able to modify the existing objects, add or delete objects from the universe, save variations of an object acquired from incomplete matches for use or merger in future experiments. In the following subsections we discuss these functions of our recognizer.

6.1 Addition or Deletion of Objects from the Universe

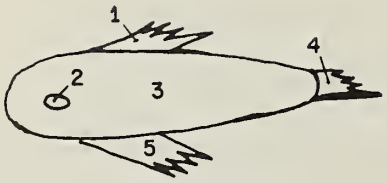
Our universe of objects (graph structure) can be easily modified to include new objects, or generalized to recognize a larger class of the same object, or add restrictions to the existing models to exclude near-miss examples of existing objects.

Introducing a new object to our universe can be accomplished in several steps. If the object is complex it should be learned part by part. In the case of simple objects all the existing rules are bound to fail, so the initial scene graph is saved as the model for that object. In Fig. 6.1(a) we have shown a fish and its graphical representation. Conceptual formation (modeling) of the new object can be formalized in the following manner:

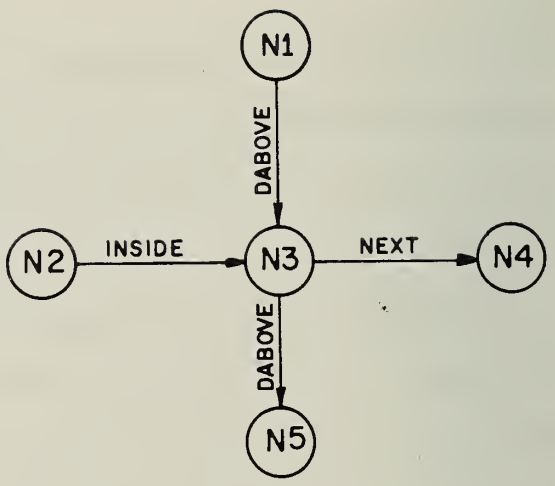
a) The input graph is parsed normally, except that the backup procedure would not be invoked. In other words, for all parts of the input graph we will try to climb in the parsing graph as high as possible.

b) The reduced graph is given a distinct rule name and saved in the graph structure.

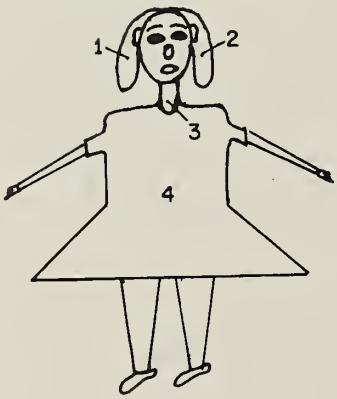
c) A node is created in the parsing graph. A set of out-going branches are created between this node and the nodes representing the graphical



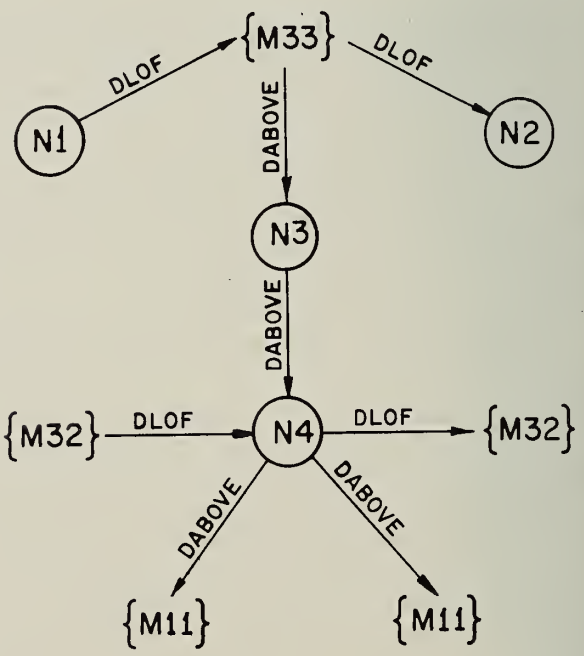
(a) fish



(b) graphical representation of fish



(c) girl



(d) final reduced graph for the girl

Fig. 6.1. Addition of new objects to the universe.

rules associated with the nodes of the newly formed graph.

d) Semantical information provided by the teacher is saved in an area pointed to by this new rule (name, object association, etc.).

This will enable our system to recognize the new object in subsequent experiments. In Fig. 6.1(c) we have shown a girl. After applying our normal parsing procedure we have identified a head (M33), two hands (M32's) and two feet (M11's). The final graph is shown in Fig. 6.1(d).

Another learning experiment for the system would be to present it with an object and tell it this is an object known to the system or this is a near-miss to an object. In both cases the final reduced graph of the object is combined with the existing model. In the first case the combined model should encompass (match) the final reduced graph. In the near-miss case the model should be modified in a manner which does not match the reduced (near-miss) input graph, but will match an already existing model.

Model generalization can be done easily by adding optional nodes and branches to the graphical rule or if necessary to make some of the existing branches optional or change their labels. For example, if a region was known to be directly above another region, but in the input scene it happened to be directly below that region, the branch would be labeled as next-to. Restriction can be easily carried out by adding new nodes and branches or make some optional elements imperative, or relabel some of the branches.

An optional element in a graph is an element which will be sought in the domain matching, but its non-existence will not affect the outcome of the match. For example, if the girl in 6.1(c) was represented as an example of a man, two optional nodes and branches will be added to the man's model (M34) and a set of attribute values will be added to the associated values

of the primitive node representing the midsection, to make the region 4 an acceptable match to this primitive node.

6.2 Saving the Incomplete Domains of the Rules

In the teaching process there are cases when we want the system to learn from the examples, but we do not provide it with additional information as to how this learning should take place. In this case the system will proceed blindly and finds out the rule which has the best-match. If this best-match is acceptable according to some criterion (AFOM, etc.), the recognizer will proceed as though the match was complete. Now, if the parsing was successful and a proper object was recognized in the scene, the domains of these best-matches are saved as variations of these best-matched graphical rules as follows:

a) The domain of the best match is saved in a graph.

b) A new node is created in the parsing graph and has this newly formed graph associated with it.

c) Create a branch between this node and the node of the best-matched rule (link) indicating that this new node represents a satellite of that rule.

Now, if with each rule we try its satellites as alternatives, we can avoid the extensive searching process for these known best matches. Using some appropriate criterion we can later on combine any of these satellites with the linked graphical rule or delete them from our graph structure. This experiment was carried out for the scene of Fig. 5.6, and recognition time was reduced from approximately 26 seconds to 16 seconds.

7. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

In this thesis we developed a general methodology for analysis and parsing of the graph representable pictures. This methodology is general in the sense that as long as the entities subject to analysis can be represented in graphs, it provides a valid technique of analysis for these entities. We have shown the application of this technique to picture analysis throughout this thesis and in particular we have demonstrated the results of its application to a simple class of pictures. Here we show its generality by listing the involved steps in any application:

1. Define the class of entities.
2. Define the primitives (atoms) of this class, which represent the primitive nodes in our graph representation.
   2.1 Define a set of primitive node attributes whose values will identify these atoms.
3. Define a class of relations which can be used to represent the contextual relationships of these atoms as well as this class of entities.
   3.1 Investigate the embedding properties of these relations, and define procedures to embed branches whenever it is required in connection with transformations.
4. Develop preprocessing techniques which can transform the natural occurrence of these entities to this graphical form.
5. Define the graph structure which represents the entities in the universe.
6. Our defined recognizer and domain-search algorithms can be applied directly.

As we have seen in chapter 6, our system has considerable learning capability. If the user chooses not to define the graph structure, he can

let the system build its own graph structure through experience and learning.

In defining the graph structure of a class of 3-dimensional objects we can choose either of the following two methods:

1. Define a graphical model for each 3-dimensional part of the objects. In this graphical representation, there should be enough information to enable us to construct the graphical representation of this part at all different projecting angles.
2. A useful alternative is the case where we allow a limited number of projecting angles. In this case we can have a single node in the model, which represents all visible regions tagged with the angles at which they are visible.

For some class of pictures (Guzman (1968)) it has been shown that there are a set of general rules, which can act on the input graph and discover the cluster of regions which correspond to these 3-dimensional parts. In this research we have chosen the second approach, and in establishing the semantic equivalence, simply all the regions visible at any angle should have a corresponding node in the scene graph.

7.1 Parallel Processing

Another area open for investigation is parallel graph processing in connection with graph structure. To achieve this parallel processing the control structure of our SOL procedures must be much more flexible than those of Pℓ/1. Our recognition system, in addition to moving up and down

hierarchical trees by initiating and terminating execution of access modules, should be able to wander among the access modules by suspending and resuming their executions in any order, unconstrained by the tree structure of their inherent control relationships. For example when two different copies of the domain-search procedure are working on two different parts of the graph it should be possible to suspend one of them from inside the other one. In [24] Bobrow and Wegbreit (1973b) define a general model for control that has been used as the basis of implementations for such languages. This model defines the set of information or frame that must be associated with every activation of an access module to make possible its suspension and reactivation in a meaningful way. The SOL graphs and associations can be used to represent this type of control structures.

7.2 Occluded Objects

In our described method the partial matching technique is useful in recognition of reasonably visible objects. Open for investigation here is the discovery of some deductive procedures which can fill in the occluded parts after the recognized object is removed. For example, in Fig. 7.1(a) after the "bird" is recognized and removed, a continuity process should be able to deduce and fill in the hidden parts of the legs. In our current implementation, the "bird" and "man" are both recognized, but the lower halves of the feet will probably be recognized as two "hammers". In Fig. 7.1(b) we have shown the input graph after the "man" and "bird" have been recognized.

In recognition of different projections of the objects, we require procedures which can project the parts of an object at any angle and match it against the part of scene which is supposed to be the projection of this part at this angle.

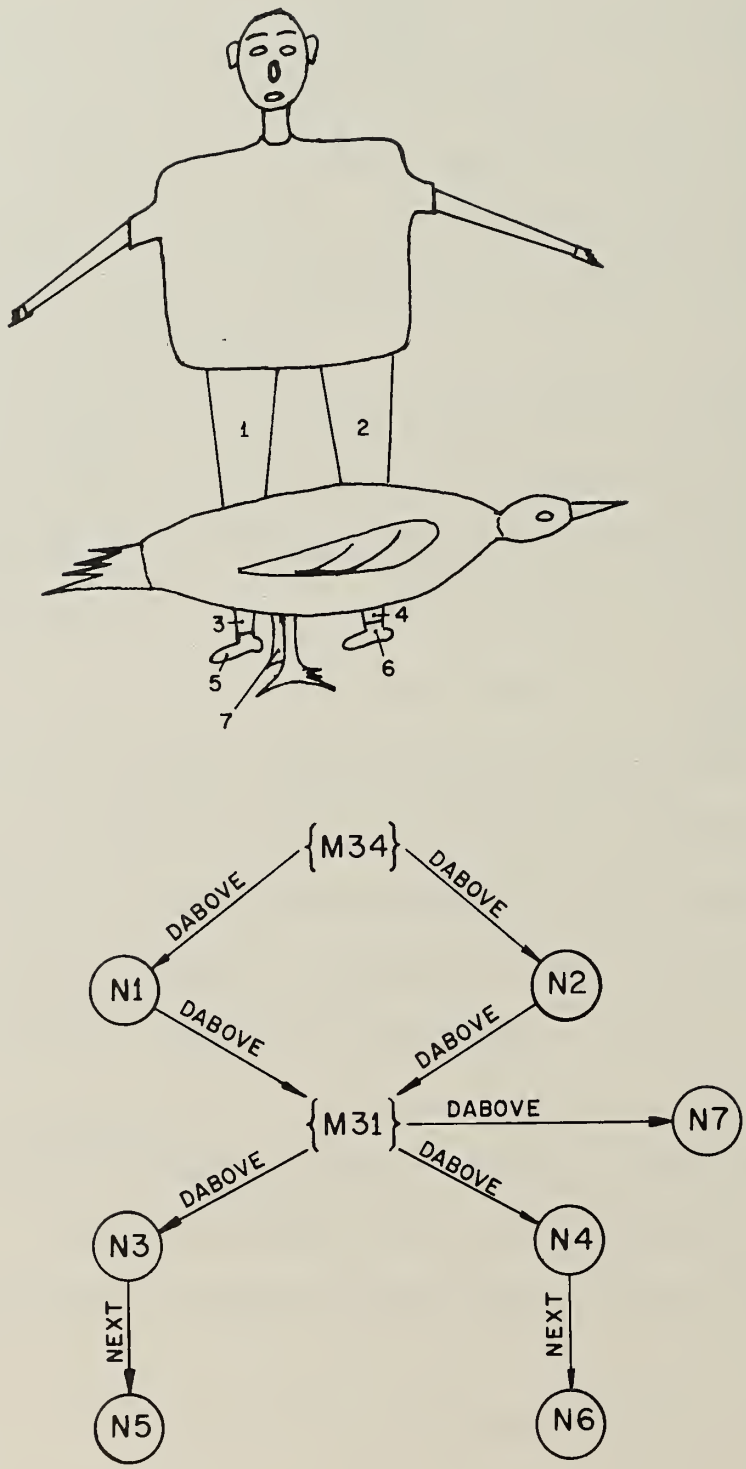


Fig. 7.1. Recognition of occluded objects.

7.3 Relational Files

In our implementation, the knowledge about the universe (graph structure) is saved exactly in the same structural form that it will be used in inferences. As the knowledge grows we will encounter the problem of partitioning this data set and managing the file. It would be desirable to have a common data base which can be referenced by our system as well as other artificial intelligence and problem solving systems. Associative memories like the one used in SAIL [25] look promising. For example

color X X = red

can be used to extract all the primitive nodes whose their "color" attribute has the value "red". The set of primitives found from the above search can be used in

father X X = Y,

to find Y, the set of successors to each primitive node. Graphs can be defined in the same manner by defining structural relations between nodes. For example,

node X G1 = X,

will locate all the nodes of graph G1. Since the graphical representations are needed for domain search and other algorithms, we must also define the interface procedures which extract this information from the data base and construct the graphs as they are needed.

We also consider the implementation of an interactive SOL imperative for further research.

LIST OF REFERENCES

[1] Uhr, L. (Ed.), Pattern Recognition, New York, Wiley, 1966.

[2] Michalski, R. S., "A Variable Valued Logic System as Applied to Picture Description", Proceedings of the IFIP, May 22-26, 1972.

[3] Maruyama, K., "A Study of Visual Shape Perception", Ph.D. Thesis, Department of Computer Science, University of Illinois, 1972.

[4] Jayaramamurthy, S. N., "Computer Methods for Analysis and Synthesis of Visual Texture", to be reported as a Ph.D. Thesis, Department of Computer Science, University of Illinois, 1973.

[5] Strong, J. P. and Rosenfeld, A., "Automatic Cloud Cover Mapping", Ph.D. Thesis, Computer Science Center, University of Maryland, 1971.

[6] Guzman, A., "Analysis of Curved Line Drawings Using Context and Global Information", in Machine Intelligence, edited by Meltzer, B. and Michie, D. (1971), pp. 325-375.

[7] Herzog, B., "Lectures on Computer Graphics", Computer and Program Organization-Fundamentals, University of Michigan Engineering Summer Conferences, June, 1967.

[8] Kursland, H. E., "A General Purpose Graphic Language", Communications of the ACM, Vol. 11, No. 4, April, 1968, pp. 247-254.

[9] Schwebel, J. C., "Towards the Specification of a New Image Processing Language", DCS File No. 788, University of Illinois at Urbana-Champaign, February, 1969.

[10] Chase, S. M., "Analysis of Algorithms for Finding All Spanning Trees of a Graph", DCS Report No. 401, University of Illinois at Urbana-Champaign, 1970.

[11] Pratt, T. W. and Friedman, D. P., "A Language Extension to Graph Processing and Its Formal Semantics", Communications of the ACM, Vol. 14, No. 7, July, 1971, pp. 460-467.

[12] Earley, J., "Toward an Understanding of Data Structures", Communications of the ACM, Vol. 14, p. 617, 1971.

[13] Lieberman, R. N., "RSVP Relational Structure Vertex Processor", Tech. Report No. 69-87, Computer Science Center, University of Maryland, March, 1969.

[14] Wolfberg, M. S., "An Interactive Graph Theory System", Report No. 69-25, Moore School of EE, University of Pennsylvania, June, 1969.

[15] Crespi-Reghizzi, S. and Morpurgo, R., "A Language for Treating Graphs", Communications of the ACM, Vol. 13, No. 5, May, 1970, pp. 319-323.

[16] Pflantz, J. L., "Web Grammars and Picture Description", Computer Graphics and Image Processing (1972), pp. 193, 220.

[17] Schwebel, J. C., "A Graph-structure Transformation Model for Picture Parsing", Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1972.

[18] Shaw, A. C., "Parsing of Graph Representable Pictures", Journal of the ACM, Vol. 17, No. 3, July, 1970, pp. 453-481.

[19] Guzman, A., "Decomposition of a Visual Scene into Three-dimensional Bodies", AFIPS Fall Joint Computer Conference, Vol. 33, 1968, pp. 291-304.

[20] Eastman, C. M., "Explorations of the Cognitive Processes in Design", Computer Science Department, Carnegie Mellon University, February, 1968.

[21] Pflantz, J. L. and Rosenfeld, A., "Web Grammars", Proceedings International Joint Conference on Artificial Intelligence, May, 1969, pp. 609-619.

[22] McCormick, B. H. and Schwebel, J. C., "Use of Graph Transformations to Characterize an Image: An Illustrative Example", File No. 770, Department of Computer Science, University of Illinois at Urbana-Champaign, July, 1968.

[23] Bobrow, D. G., "Natural Language Input for a Computer Problem Solving System," Semantic Information Processing, MIT Press, pp. 135-215.

[24] Bobrow, D. G. and Wegbreit, B., "A Model for Control Structures for Artificial Intelligence Programming Languages", Proceedings of IJCAI, Stanford, California, 1973.

[25] Feldman, J. A., et al., "Recent Developments in SAIL", An ALGOL-based language for artificial intelligence, FJCC, 1972.

[26] Rosenfeld, A., "Picture Processing", Computer Graphics and Image Processing, 1972, pp. 394, 416.

[27] Advani, J. G., "Computer Recognition of Three Dimensional Objects from Optical Images", Ph.D. Thesis, Ohio State University, 1972.

[28] Gaffney, J. L., Jr., "TACOS: A Table Driven Compiler System", DCS Report No. 325, University of Illinois at Urbana-Champaign.

[29] Bobrow, D. G., "New Programming Languages for AI Research", Third International Joint Conference on Artificial Intelligence, Stanford University.

[30] Preparata, F. P. and Ray, S. R., "An Approach to Artificial Non-Symbolic Cognition", Information Sciences, 4 (January, 1972), pp. 65, 86.

[31] Buneman, O. P., "A Grammar for the Topological Analysis of Plane Figures", Machine Intelligence 6, ed. by Meltzer and Michie, Edinburgh University Press, 1970.

[32] McCormick, B. H., "Experiments with an Image Processing Computer", University of Illinois, DCS File No. 406, June, 1970.

[33] Stanton, R. B., "Plane Regions: A Study in Graphical Communication", DEC, University of South Wales, pp. 151, 193.

[34] Zahn, C. T., "Graph-theoretical Methods for Detecting and Describing Gestalt Clusters", IEEE Transactions on Computers, SLAC-PUB-672, Nov., 1969.

[35] Schwebel, J. C., "Graph Transformations for Composite Formations", DCS Report No. 368, University of Illinois, December, 1969.

[36] McCormick, B. H. and Schwebel, J. C., "Consistent Properties of Composite Formation under a Binary Relation", DCS Report No. 348, University of Illinois, August 1969.

[37] Love, H. H., Jr. and Savitt, D. A. and Troop, J. R. E., "ASP: A New Concept in Language and Machine Organization", SJCC, 1967.

[38] Michalski, R. S., "A Geometrical Model for the Synthesis of Internal Covers", DCS Report No. 461, June, 1971, University of Illinois.

[39] McCormick, B. H. and Michalski, R. S., "Interval Generalization of Switching Theory", University of Illinois, DCS Report No. 442, May, 1971.

[40] Shaw, A. C., "On the Interactive Generation and Interpretation of Artificial Pictures", SLAC-PUB-664, Cornell University, Ithaca, New York.

[41] Smith, D. N., "GPL/1-APL/1 Extention for Computer Graphics", SJCC, 1971, pp. 511-528.

[42] Anderson, J., Balke, K. G., and Earnest, C. P., "Analysis of Graphs by Ordering of Nodes", Journal of the ACM, Vol. 19, No. 1, January, 1972.

[43] Firscheing, O. and Fischler, M. A., "Describing and Abstracting Pictorial Structures", Pattern Recognitions, 1971, pp. 421, 443.

[44] Winogrud, "Cognitive Psychology", Vol. 3, No. 1, whole issue.

[45] Fu, K. S. and Swain, P. H., "On Syntactic Pattern Recognition", Software Engineering, Academic Press, 1971.

[46] Evans, T. G., "Grammatical Inference Techniques in Pattern Analysis", Software Engineering, Academic Press, 1971.

[47] Rosenfeld, A. and Strong, J. P., "A Grammar for Maps", Software Engineering, Academic Press, 1971.

[48] Minsky, M., "Semantic Information Processing", The MIT Press, 1968, whole book.

[49] Pavlidis, T., "Linear and Context-free Graph Grammars", Journal of the ACM, Vol. 19, No. 1, January, 1972, pp. 11, 22.

[50] Winston, P. H., "Learning Structural Descriptions from Examples", MAC-TR-76, Project MAC, MIT, September, 1970.

[51] Narasimhan, R., "On the Description, Generation, and Recognition of Classes of Pictures", Automatic Interpretation and Classification of Images, New York and London: Academic Press, 1969.

[52] Clowes, M. B., "Transformational Grammars and the Organization of Pictures", Automatic Interpretation and Classification of Images, New York and London: Academic Press, 1969.

[53] Evans, T. G., "Descriptive Pattern Analysis Techniques", Automatic Interpretation and Classification of Images, New York and London: Academic Press, 1969.

[54] Berkowitz, S., "GIRL-Graph Information Retrieval Language--Design of Syntax", Software Engineering, Vol. 2, Academic Press, 1971.

[55] Dodd, G. G., "APL--A Language for Associative Data Handling in PL/1", FJCC (1969), pp. 677-684.

[56] Knowlton, K. C., "A Programmer's Description of L^6", ACM, Vol. 9, 1966, pp. 616-625.

[57] Rovner, P. D., "An AMBIT/G Programming Language Implementation", MIT, June, 1968.

[58] Feldman, J. A. and Rovner, P. D., "The Leap Language and Data Structure", Information Processing, pp. 579-585.

[59] Pavlidis, T., "Structural Pattern Recognition: Primitives and Juxtaposition Relations", Princeton University, January, 1971.

[60] Nagy, G., "State of the Art in Pattern Recognition", Proceedings of the IEEE, Vol. 56, 1968, pp. 836,862.

[61] Levine, M. D., "Feature Extraction: A Survey", Proceedings of the IEEE, Vol. 56, 1968, pp. 836-862.

[62] Falk, G., Feldman, J. A., and Paul, R., "The Computer Representation of Simply Described Scenes", in M. Faiman and J. Nievergelt (Eds.), Pertinent Concepts in Computer Graphics, University of Illinois Press, 1969, pp. 87, 103.

[63] Anderson, R. H., "Syntax Directed Recognition of Handprinted Two-dimensional Mathematics", Ph.D. Thesis, Harvard University, January, 1968.

[64] Feder, J., "Linguistic Specification and Analysis of Classes of Line Patterns", School of Engineering, EE Department, New York University, April, 1969.

[65] Feder, J., "The Linguistic Approach to Pattern Analysis: A Survey", Report No. 400-133, School of Engineering, EE Department, New York University, February, 1966.

[66] Kirsch, R. A., "Computer Interpretation of English Text and Picture Patterns", EC-13, No. 4, IEEE Trans. on Electronic Computers, August, 1964, pp. 363, 376.

[67] Knoke, P. J. and Wiley, R. C., "A Linguistic Approach to Mechanical Pattern Recognition", Proceedings of the IEEE Computer Conference, September, 1967, pp. 142-144.

[68] Barrow, H. G. and Popplestone, R. J., "Relational Description in Picture Processing", in Meltzer, B. and Michie, D. (Eds.), Machine Intelligence 6, Edinburgh University Press, 1971, pp. 325, 375.

[69] Eastman, C. M., "Explorations of the Cognitive Processes in Design", Computer Science Department, Carnegie-Mellon University, February, 1968.

[70] Eastman, C. M., "Representations for Space Planning", Communications of the ACM, Vol. 13, No. 4, April, 1970, pp. 242, 250.

[71] Montanari, G. U., "Networks of Constraints: Fundamental Properties and Applications to Picture Processing", Department of Computer Science, Carnegie-Mellon University, January, 1971.

[72] Duda, R. O. and Hart, P. E., "A Survey of Pattern Classification and Scene Analysis", Stanford Research Institute, January, 1971.

[73] Huffman, D. A., "Impossible Objects as Nonsense Sentences", in Meltzer, B. and Michie, D. (Eds.), Machine Intelligence 6, Edinburg University Press, 1971.

[74] Pavlidis, T., "Computer Recognition of Figures through Decompositions", Information and Control, Vol. 13, 1968, pp. 526, 537.

[75] Pavlidis, T., "Computer Analysis of Figures into Primary Convex Subsets", Record of the IEEE SSC Conference, cat. no. 68C23-SSC, 1968, pp. 55-60.

[76] Salton, G. and Sussengath, E. H., "Some Flexible Information Retrieval Systems Using Structure Matching Procedures", AFIPS, Proceedings of the Spring Joint Computer Conference, 1964, pp. 587, 597.

[77] Matula, D. W., "Cluster Analysis via Graph Theoretic Techniques", in Mullin, R. C., Reid, K. B. (Eds.), Proceedings of the Louisiana Conference on Graph Theory Combinatorics and Computing, 1970, pp. 199, 212.

[78] Zahn, C. T., "Graph-theoretical Methods for Detecting and Describing Gestalt Clusters", IEEE Transactions on Computers, Vol. C-20, No. 1, January, 1971, pp. 68, 86.

[79] Pflantz, J. L., "Convexity in Graphs", TR-68-74, Computer Science Center, University of Maryland, July, 1968.

[80] Montanari, G. U., "Separable Graphs, Planar Graphs, and Web Grammars", Information and Control, Vol. 16, No. 3, May, 1970, pp. 243, 267.

[81] Evans, T. G., "A Heuristic Program to Solve Geometric-Analogy Problems", AFIPS, Proceedings of the Spring Joint Computer Conference, 1964, pp. 327, 328.

[82] Siklossy, L., "Generalized Means-Ends Analysis and Artificial Intelligence", Information Sciences, Vol. 3, 1971, pp. 149, 158.

[83] Holden, A. D. C. and Johnson, D. L., "The Use of Embedded Patterns and Canonical Forms in a Self-Improving Problem Solver", Proceedings of the ACM National Meeting, 1967, pp. 211, 219.

[84] Winston, P. H., "A Heuristic Program that Constructs Decision Trees", AI Memo 173, Project MAC, MIT, March, 1969.

APPENDIX A

SOL

Various programming languages are specialized in the sense that the set of unique facilities provided by a language makes some types of programs easier to write in that language than in any other. Indeed, the main reason for introducing new features into a programming language is to automate procedures that the user needs and would otherwise have to code explicitly; such features reduce the housekeeping details that distract the user from the algorithms in which he is really interested. The structure operation language (SOL) has been designed and implemented to facilitate the representation and transformation of graph structures and associated necessary operations in the field of artificial intelligence, specifically in picture processing. SOL consists of statements called graph structure statements which are embedded in the procedural language Pℓ/1. SOL statements consist of declarations, associations and operations.

Declarations declare the basic structural elements. Associations declare and associate any Pℓ/1 data type (excluding based and controlled variables) with the basic structural elements. Operations are performed on the declared structural elements, and are capable of dynamically creating or deleting elements.

Basic Structural Elements

There are six basic structural elements which are designated by the attributes in the declarations or implied by the class of operations which create them dynamically. These elements are pointer, set, node, branch, subgraph and graph.

A pointer points to or designates any other basic element. A pointer can designate a pointer, set, node, branch, subgraph or graph. Thus, a pointer allows indirect reference to any element. SOL pointers are useful in forming a variable set of elements; in other words, although the elements of the set are the same set of pointers, we can effectively change the set by reassigning different values to these pointers.

A set designates a set of basic elements. These elements can include SOL pointers, which in turn can point to any other basic structural elements.

Nodes, branches and subgraphs are elements of graphs. A node designates a set of branches which are called adjacent branches of the node. A branch designates a pair of nodes. An oriented branch designates an ordered pair of nodes, the tail and the head of the branch. An unoriented branch designates an unordered pair of nodes.

A graph is a set of nodes, branches and subgraphs with the property that if a branch belongs to a graph then its tail and head also belong to the graph. Graphs will be assumed to have oriented branches unless specifically declared otherwise.

A subgraph is a collection of nodes and branches such that if a branch belongs to the subgraph its head and tail must also belong to the subgraph. A collection of disjoined subgraphs is such that no node is shared in common with other subgraphs of the collection.

Declarations

The DECLARE statement is used to declare the six basic structural elements. An element is declared by an identifier followed by an attribute which specifies the type of that element.

Declaring an element creates a new data element with the name given by the identifier. A level hierarchy is assumed within a declaration such that

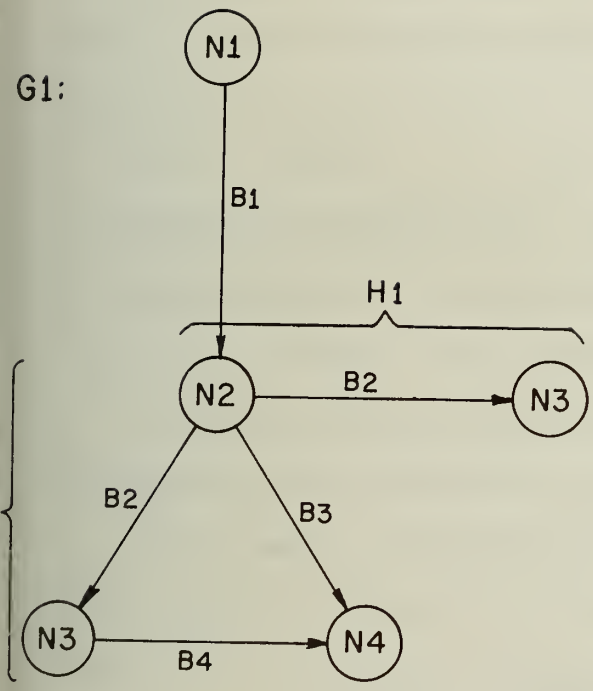
any graph element declared is assumed to belong to the last declared element (graph or subgraph) at a lower numbered level. Nodes and subgraphs in the scope of a disjoint subgraph (or graph) are assumed to have a unique name. The repetition of the same name in the same scope is interpreted as a reference to the same element. The default value for graph level is 1, and it is 2 for all other elements of the graph. A branch is always identified by the end nodes, so the declaration of a branch with the same tail and head and the same name is interpreted as a reference to an existing branch.

The attribute of a branch is of the form BR(N1,N2), where N1 and N2 are names of tail and head of the branch respectively. This declaration will also declare the head and tail as nodes if they have not already been declared. Fig. A.1 shows some examples of declarations. The default value for the subgraph is non-disjoint.

Names and References to Elements

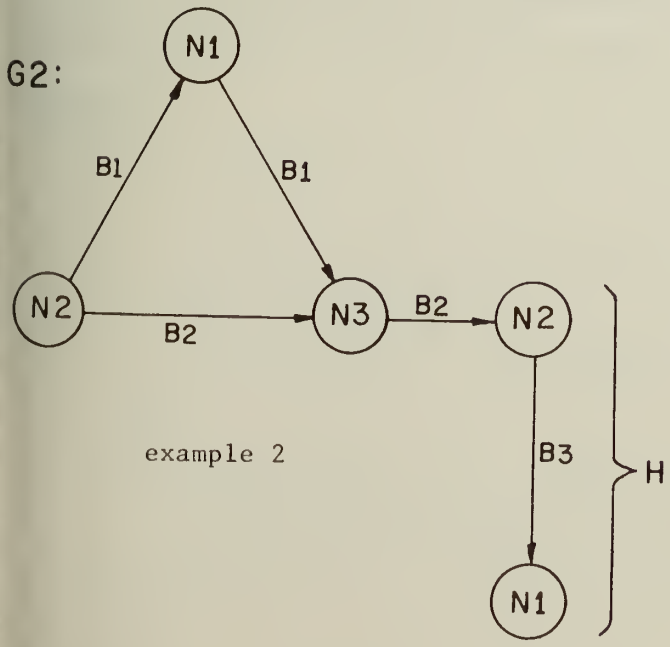
Names are not required for nodes and branches even though they are explicitly required in the DECLARE statement. Elsewhere in the SOL language, operations which dynamically modify a structure can result in the creation of a new unnamed element. Basically names need not be unique, since the basis of references are pointers pointing to different elements of the graph. Elements with non-unique names may be referenced uniquely, for instance by their position relative to other elements.

By a reference to an element, we mean a unique way of referring to that element. An element can be referenced by its name when the name is unique within the current scope or context. In Fig. A.1 example 2, nodes with the name N2 can be referenced uniquely by N2 and H.N2 respectively. Reference by name is important in the process of declaration. If we wish two nodes or



example 1

```
DCL
   G1 GR,
    2 N1 ND,
    2 B1 BR(N1,N2),
    2 H1 SG,
       3 B2 BR(N2,N3),
    2 H2 SG,
       3 B2 BR(N2,N3),
       3 B3 BR(N2,N4),
       3 B4 BR(N3,N4);
```



example 2

```
DCL
   G2 GR,
      B1 BR(N1,N3),
      B1 BR(N2,N1),
      B2 BR(N2,N3),
    2 H  SG DISJOINT
       3 B3 BR(N2,N1),
    2 B2 BR(N3,H.N2);
```

Fig. A.1. Declaration examples.

subgraphs to have the same name in the same scope we can given them temporary names, and change these names in subsequent operations. Since branches, in addition to name, are identified with their tail and head nodes, they need not to have unique labels.

An element can always be referenced by a pointer, since a pointer always points to only one element. Names for graphs, pointers and sets are always references, i.e. they are unique in the current scope. Subgraphs, nodes and branches, however, may have nonunique names. In this case, they may be referenced by a pointer, by a qualified name, or by other contextual means. The nondisjoint subgraphs need not be used to qualify their elements.

General format

```
declaration:: = {DECLARE|DCL}[integer]⟨*I⟩ attribute
                  {,[integer]⟨*I⟩ sttribute}* ;
attribute:: = {GRAPH|GR}[UNORIENTED]|
              {SUBGRAPH|SG}[DISJOINT]|
              {NODE|ND}|
              {BRANCH|BR}({⟨*I⟩|node-ref},{⟨*I⟩|node-ref})|
              {SET|ST}|
              {POINTER|PT} .
```

Associations

An ASSOCIATE statement allows PL/1 variables (excluding controlled and based variables) to be declared and associated with the specified set of basic structural elements of a given type. If the set is not specified, it will be associated with all the elements of that type. The FREE statement nullifies the association. Thus, we can think of an association as function, which takes elements as its argument, and returns the value of the associated variable.

The list of element references specifies the elements (or the set of elements) to be associated with or freed from the variables which follow. Examples:

```
BRASOC   LENGTH   BIN   FIXED;
NDASOC   (G1.ND3,NOF(G2))   ARRAY(10)DEC   FIXED,
                            ANGEL   BIN   FIXED,
                            TEXT   CHAR(30);.
```

The first example will associate a variable "LENGTH" of type FIXED BINARY with all the branches of all the known graphs in the current scope. The second statement will associate an array of 10 elements of type DEC FIXED, a variable "ANGEL" of type FIXED BINARY, and a character string "TEXT" 30 characters long, with node "ND3" of graph "G1" and all the nodes of graph "G2".

Since the based variables are used to implement associations, all restrictions to based variables apply to the associated variables in SOL. The value of a variable associated with an element is accessed by using the variable name followed by an element reference enclosed in parentheses. These associated variables can be used as pseudo-names to assign values. Examples:

```
LENGTH(G1.B1),
LENGTH(P1),  /*P1  pointer to a branch */
ARRAY(G1.ND3)(5),
ANGEL(TAIL(P2)),  /*P2 pointer to a branch in graph G2*/
```

syntax;

```
association:: = associate|free
associate:: = { PTASOC|STASOC|NDASOC|BRASOC|SGASOC|GRASOC }
              [(element-ref{,element-ref}*)]PL1-declaration-tail;
```

```
Pℓ1-declaration-tail:: = Pℓ1-element-dcℓ{,Pℓ1-element-dcℓ}*

free:: = {PTFREE|STFREE|NDFREE|BRFREE|SGFREE|GRFREE}
         [(element-ref{,element-ref}*)]⟨*I⟩{,⟨*I⟩}* ;
```

examples:

```
GRASOC(G1)   1   A,
                 2   B   BIN   FIXED,
                 2   C   CHAR   (8) ;

GRFREE(G1)   A;
```

Data Operations

Data operations dynamically add or delete elements to or from previously declared graphs, subgraphs and sets. A reference to the element being modified appears after the operation name. The identifier and attributes (or context) which follow refer to new elements to be added or elements to be deleted.

Deleting an element from a set does not destroy the element but only deletes the reference to the element contained in the set. Similarly, deleting a node or branch from a non-disjoint subgraph will not destroy the node or branch, since it will still be a member of a graph or a disjoint subgraph at a lower level. Deleting an element from a graph or a disjoint subgraph, however, will delete the element from existence.

Examples:

```
ADD NODE N5 TO GR G1; /*adds node N5 to graph G1*/

DEL NODE N5 FROM G1.H; /*deletes node N5 from subgraph H*/

DEL P1 FROM SET S1; /*deletes pointer P1 from set S1*/

ADD SUBGRAPH SG1 TO GR G2  /*adds subgraph SG1 to graph G2*/

ADD BR NEXT (N1,N2) TO G2; .
```

/* adds a branch with label "NEXT" between nodes N1 and N3 of graph G2*/.
Deleting a node, a branch or a subgraph will also delete it from the sets referring to this element.

Syntax:

data-operation:: = add-operation|delete-operation

add-operation:: = ADD element-ref TO element-ref;

delete-operation:: = {DEL|DELETE} element-ref FROM element-ref; .

We will define element-ref later. Some combinations of the two element-ref's appearing in data-operation are not acceptable.

Loop-Control

The loop-control statement allows the iteration of the statements between "FOR" statement and corresponding "END" statement for all the elements of the set specified in this statement. For (i,v,S) specifies that each iteration will have a different value of the variable v chosen from the set S. The variable i specifies the number of iterations to be performed. "ANY" is equivalent to 1 and "ALL" is equivalent to the cardinality of the set S.

Changing the set S within the loop can change the number of iterations performed. For example, if an element x is deleted from s before x is executed as the value of the loop-control variable v, then x will not be used.

Example:

FOR (ALL,NP,NOF(G));···END; .

The loop-control is a very useful statement.

Syntax:

loop-control:: = FOR({ANY|ALL|integer}, pointer-ref,set-ref);|END; .

Element-operations

These operations operate on a single or a pair of elements and return a value depending on the nature of the operation.

Sets-set

Sets-set are binary operations on sets which return a set. The returned set is the UNION, INTERSECTION, DIFFERENCE, or SYMMETRIC DIFFERENCE of the given sets. If the two arguments are respectively S_1, S_2 then,

$$\text{UNION}(S_1,S_2) \equiv S_1 \cup S_2 \quad \text{—— OR}$$

$$\text{INTER}(S_1,S_2) \equiv S_1 \cap S_2 \quad \text{—— AND}$$

$$\text{DIF}\ (S_1,S_2) \equiv S_1 - S_2 \quad \text{——} \ S_1 \cap \overline{S_2}$$

$$\text{SYMDIF}(S_1,S_2) \equiv S_1 \oplus S_2 \quad \text{——} \ S_1 \cap \overline{S_2} \cup \overline{S_1} \cap S_2 \ .$$

Example:

S = UNION(S_1,BOF(G));

which assigns to set S the union of set S_1 and set of all branches in graph G.

Syntax:

sets-set:: = set-set-mnemonic (set-ref,set-ref)

set-set-mnemonic:: = UNION|INTER|DIF|SYMDIF .

Sets-Boolean

Sets-boolean returns a boolean value corresponding to the truth or falsity of the statement;

"S_1 equals S_2",

"S_1 is a subset of S_2",

Syntax:

sets-boolean:: = {EQUALS|SUBSET}(set-ref,set-ref) .

Graph-set

The graph-set operates on a graph and returns the set of nodes or branches of the graph.

Example:

```
S = NOF(G1)  /* S is the set of all nodes in graph G1 */ .
```

Syntax:

```
graph-set:: = {NOF|BOF} ({graph-ref|subgraph-ref})
graph-ref:: = [GRAPH|GR]⟨*I⟩|pointer-ref .
```

Node-set

These operations operate on a node and return the set of incoming (INCBR), outgoing (OUTBR) or adjacent (ADJBR) branches of the node, respectively.

Example:

```
S = ADJBR(G1.N1);
```

Syntax:

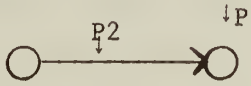
```
node-set:: = {ADJBR|INCBR|OUTBR}(node-ref)
node-ref:: = branch-node|[NODE|ND] qualified-name|pointer-ref .
```

Branch-node

The HEAD and TAIL functions operate on a branch and return the head and tail of the branch respectively.

Example:

```
P1 = HEAD(P2) ;
```



.

Syntax:

```
branch-node:: = {HEAD|TAIL} (branch-ref)
branch-ref::  = [BRANCH|BR]⟨*I⟩(node-ref,node-ref)|[BRANCH|BR]
                qualified-name|pointer-ref .
```

Set-integer

The CARD function operates on a set and returns an integer value which indicates the cardinality of the set.

Example:

```
N = CARD(NOF(G));
N = CARD(UNION(S1,S2));
```

syntax:

```
set-integer:: = CARD (set-ref) .
```

Set-ref

Set reference is one of the following: sets-set, node-set, graph-set or a name-reference (PL/1 pointer).

Syntax:

```
set-ref:: = sets-set|node-set|graph-set|[SET|ST]⟨*I⟩ .
```

NAME and TYPE

The "NAME" function returns the name of the referenced element. The "TYPE" function returns the type of the element referenced. The types are "PT", "ST", "ND", "BR", "SG", and "GR".

The "NAME" and "TYPE" functions can also be used as pseudo variables to alter the name and type of the elements respectively.

Syntax:

```
element-string:: = {NAME|TYPE}(element-ref) .
```

The following is a summary syntax of element-operations:

```
element-operations:: = sets-set|boolean-set|graph-set|node-set
                       |branch-node|set-integer|element-string .
```

Important

When we wish to use a SOL operation in a PL/1 statement an @ should be added to the name of the operation and the PL/1 pointer used as element

references. It is obvious that this PL/1 pointer should be initialized to the proper value before it is used.

Element-ref

An element reference refers to any of the six basic elements in SOL. Syntax:

element-ref:: = pointer-ref|set-ref|node-ref|branch-ref
|subgraph-ref|graph-ref

pointer-ref:: = [POINTER|PT]⟨*I⟩

subgraph-ref:: = [SUBGRAPH|SG] qualified-name|pointer-ref.

Others have already been defined.

Pointer-operations

Pointer-operations change the value of a pointer. They are primarily useful for moving a pointer to adjacent nodes and branches of the current element pointed to, or move the pointer to any other element of the graph which has the same type. This is also necessary when we have to use contextual information for reference.

The "MOVE" operation moves the pointer to an adjacent branch if it points to a node, or to an adjacent node if it points to a branch. If the node (branch) which we are moving to is specified by name or reference, the pointer can only be moved to a node (branch) with that name or reference. If the named node (branch) does not exist as an adjacent node (branch), the pointer is not changed.

The "OMOVE" or oriented move operation is similar to "MOVE" except that moves will be made along branches only in the tail-to-head direction.

The "JUMP" operation moves the pointer to any node or branch on the graph. "JUMP" will only move from a node to a node or from a branch to a branch, so that the type of element pointed to is not changed. If no

reference to the new element is present, an arbitrarily picked node (branch) is used. If a reference by name exists and no node (branch) with that name can be found, then the pointer remains unchanged.

The "MOVE2" operation is similar to two consecutive moves, so that a node pointer is moved to an adjacent node or a branch pointer is moved to an adjacent branch. "MOVE2" is not equivalent to two consecutive "MOVEs", since the operation will be performed completely or not at all. Thus for example MOVE2 PTR L LEFT-OF L CUBE; will move a pointer named PTR along a branch (from the current node) named "LEFT-OF" to a node named "CUBE", only if both the branch and the node exist. However, the sequence MOVE PTR L LEFT-OF; MOVE PTR L CUBE; can result in only moving the pointer to a branch named "LEFT-OF". If "CUBE" was not specified, PTR is moved via branch "LEFT-OF" to an adjacent node. If "LEFT-OF" was not also specified, then PTR is moved via any branch to an adjacent node. If the specified conditions are not satisfied the pointer remains unchanged.

The "OMOVE2" is like MOVE2, except the fact that moves are made along the branches from tail-to-head direction only.

Assume a pointer named "BUG" is currently pointing to a node. Then the operations:

```
MOVE BUG L B1; /* moves BUG to an adjacent branch named "B1"*/
MOVE BUG; /* moves BUG to an adjacent branch */
MOVE2 BUG; /* moves BUG to an adjacent node */
MOVE2 BUG L ABOVE; /* moves BUG along a branch named "ABOVE" to
                      an adjacent node */
OMOVE2 BUG L HIGHER PT PTR;/* will move the BUG along an outgoing
                      branch named "HIGHER" to a node referenced by pointer
                      PTR*/ .
```

Syntax:

```
pointer-operation:: = {JUMP|MOVE|OMOVE}pointer-ref[{branch-ref
                      |node-ref}|label-name]
                      |{MOVE2|OMOVE|}pointer-ref[{branch-ref|label-name}
                      [{node-ref|label-name}]|{node-ref|label-name}
                      [{branch-ref|label-name}]];
label-name:: = {LABEL|L}⟨*I⟩ .
```

Higher-level graph operations

Node-operations and subgraph-operations deal with a single node and a collection of nodes in the graph, respectively.

Node operations

These operations are generative. PARTITION can operate on a node and partition it into a set of nodes with branches among them. Normally a graph or subgraph reference will tell us how the new elements are to be generated. The creation of branches between the nodes of this created subgraph and other nodes of the graph are application dependent and must be specified in a user prepared SOL program. This procedure can be referenced by name in the partition statement.

The operation GENERATE is like PARTITION, but it preserves the partitioned node and maintains a link between the node and generated subgraph. Example:

```
PARTITION   PTR1 INTO PTR2  AS PROC1; .
```

Here PTR1 is pointer reference to a node, and PTR2 points to a graph or a subgraph, while procedure "PROC1" dictates the policy of how this (graph or subgraph) should be placed in the original graph.

Syntax:

node-operation:: = {PARTITION|GENERATE}node-ref[INTO]
{graph-ref|subgraph-ref} AS ⟨*I⟩; .

Subgraph-operations

The MERGE operation is the inverse of PARTITION, and it reduces (merges) a collection of nodes and the branches between them (a subgraph) into a single node. The new node may be named or may be pointed to by assigning the result of the merge (new node) to a pointer. In our implementation a pointer to the new node is always assigned to the PL/1 pointer "$ELPTRØ".

Since parts of the merge operation are application dependent, the user should provide his own procedure to handle these case dependent parts.

Following actions occur when nodes XI are merged into a node X:

1. A new node, X, is created.
2. The user specified procedure will embed the necessary branches and make necessary associations with the newly formed node.
3. All nodes of the subgraph and all branches adjacent to these nodes are deleted.

The PARSE operation is similar to the MERGE operation, except that the merged subgraph is not deleted, but only the type of nodes and their adjacent branches in this subgraph are changed to make them temporarily inactive. Again a user provided procedure would be necessary to embed necessary branches and process associations. A list of these merged nodes is kept as the memory of the new node. Preserving this information makes the parse (transformation) reversible.

The BACKUP operation reverses the actions taken by parse operation and the environment is restored to the state before the parse was applied.

Example:

```
BACKUP   PT   PTR;
```

will restore the graph to the state before the parse created the node pointed to by PTR. If the memory of this node was null nothing would happen.

The DISCONNECT operation disconnects all branches between any node outside and any node inside the set of nodes specified. If a label is specified only branches with that name are deleted.

The SINK and SOURCE operate like DISCONNECT, except that SINK does not disconnect branches directed into the node set, and SOURCE does not disconnect branches directed out of the node set.

The operation LINK links all nodes into an arbitrary ordered chain by branches having the specified name or no name.

Examples:

```
MERGE   NOF(SG G1.H); /* reduce all the nodes in subgraph H into a
                         single node */
PARSE   SET   SET2 ; /*PARSE node set "SET2"*/
SOURCE ST  SET1  L  CUT; /* cut all branches with label "CUT" which are
                         directed to any node in set SET1 */ .
```

Syntax:

```
subgraph-operation:: = BACKUP node-ref|{MERGE|PARSE|
                       DISCONNECT|SINK|SOURCE|LINK}
                       set-ref[{label-name|branch-ref}][AS(*I)];
```

some of these operations are shown in Fig. A.2.

Other statements

There are a score of other statements (IF-statement, GO TO, procedure CALL, assignment, etc.) which have been implemented in this first compiler

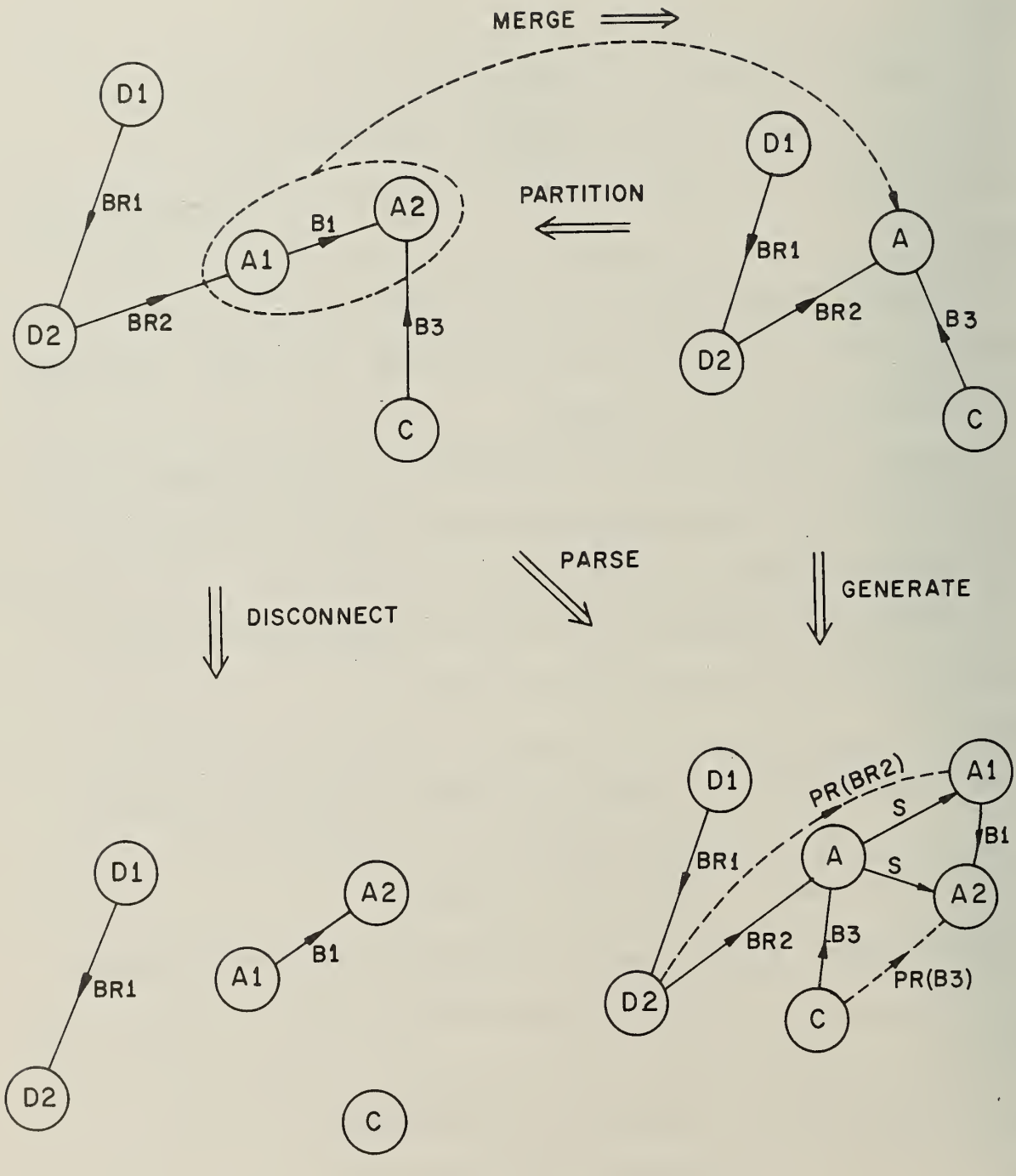


Fig. A.2. Graphical representation of some higher level operations.

for SOL, to make it a reasonably self-contained language. Most of these statements have the same syntactical and semantical interpretation as their corresponding ones in PL/1. Here we discuss only assignment statements.

Assignments

The assignments are mainly used to assign the values to the variables associated with the SOL elements and restore these values. They are also used to initialize sets and assign the results of element-operations in SOL to PL/1 variables.

Examples:

```
LENGTH(PT PTR) = 3.5; /*PTR is a pointer to a branch */
PTR1 = NODE G1.ND1
ANGEL(PTR1)(2) = 90; /*ANGEL is an array variable associated with
                       node ND1 of graph G1 */
PTR2 = ADJBR(TAIL(G1.BR1));
S1 = (PTR1, NODE G1.N3,PTR3); /*initializes S1 to the set of 3 nodes */.
```

There is another type of assignment which becomes useful when the hierarchical structure of a graph is too deep. This can facilitate the programming by allowing defined variables to be used in place of qualified-name to refer to an element in the graph.

Example: Assume the qualified name for a node is G1.H1.H2.H3.ND1, and the elements in subgraph H_3 are used several times in the program,

```
#AUX = G1.H1.H2.H3;
```

makes the qualified name as simple as #AUX.ND1 and can be used with all other elements in subgraph H_3.

Syntax:

```
assignment:: = set-assignment|equ-assignment|assoc-assignment
                |nor-assignment
```

```
set-assignment:: = set-ref = (element-ref{,element-ref}*);

equ-assignment:: = #⟨*I⟩ = qualified-name;

assoc-assignment:: = qualified-name (element-ref)
                        [(P𝓁1-index{,P𝓁1-index}*)] =
                    {qualified-name(element-ref)[(P𝓁1-index{,P𝓁1-index}*)]
                        |P𝓁1-expression};

nor-assignment:: = ⟨*I⟩[(P𝓁1-index{,P𝓁1-index}*)] =
                    {element-operation|qualified-name(element-ref)
                    [(P𝓁1-index{,P𝓁1-index}*)]|type qualified-name|P𝓁1-expression};

type:: = {POINTER|PT}|{SET|ST}|{NODE|ND}|{BRANCH|BR}
                        |{SUBGRAPH|SG}|{GRAPH|GR} .
```

Some remarks on implementation

The implementation of the data-structure for SOL will be most efficient in storage space, while meeting the requirements for dynamic addition and deletion of arbitrary elements, if the representation is a list structure corresponding topologically to the actual element structure.

Figure A.3 shows the current implementation for the basic structural elements of SOL. Each element has a unique entry in the Element Table (ETAB). The entry has a name field (8 characters), a type field (2 characters), and 4 pointer fields. Only two of these pointer fields are shown in this figure. The first pointer points to the list which defines data structures associated with this element. The second pointer will always point (possibly through a chain of pointers) back to elements in the ETAB. Since nested sets are allowed, the type field is necessary to determine the terminal elements (nodes, branches and subgraphs) of a graph or a set. The third pointer (not shown in the figure) is used to point back to the element which has this

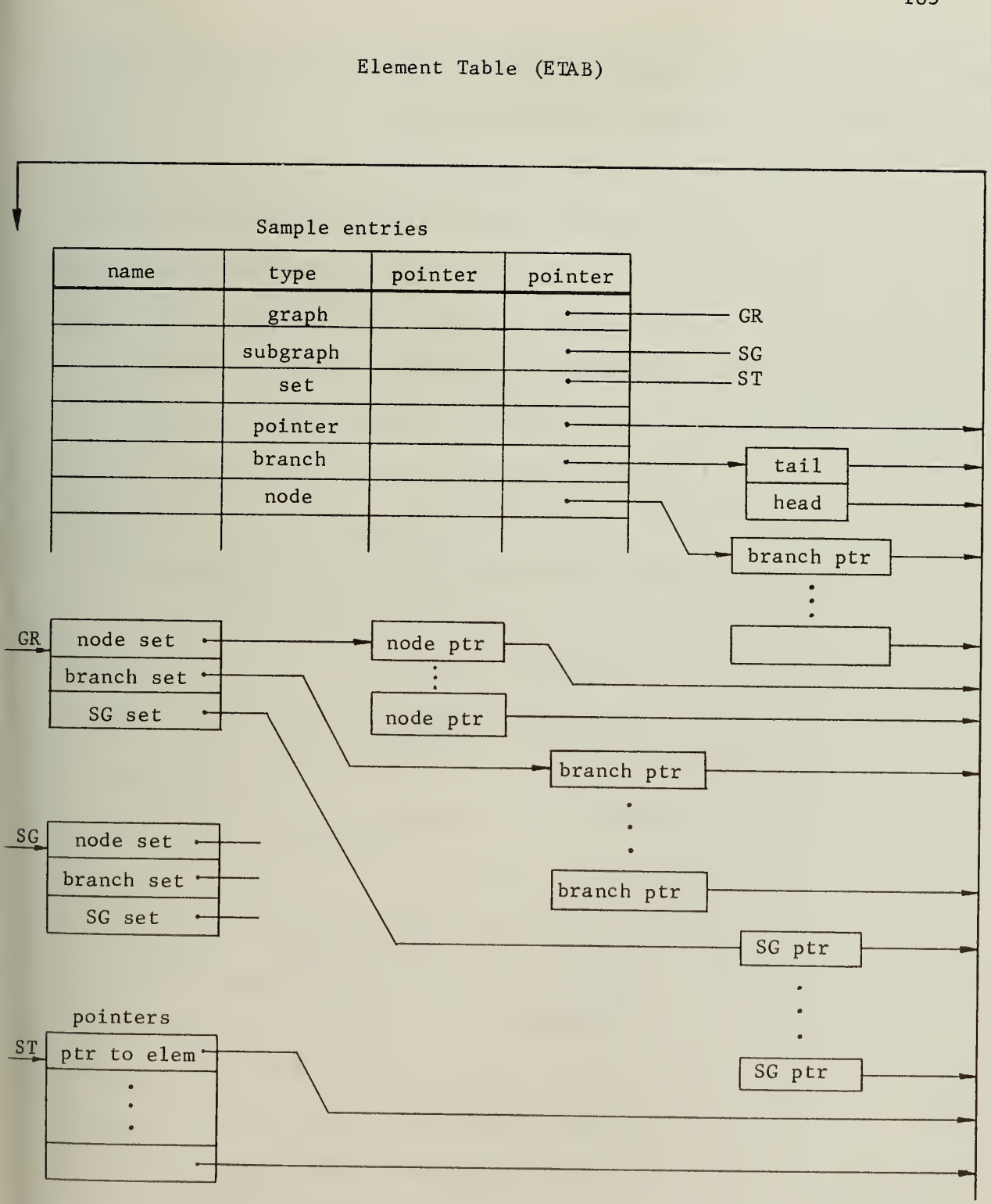


Fig. A.3. SOL data-structure.

element as one of its constituents. The fourth and last pointer is used to save the memory of this element (nodes and branches).

The sample entries in the figure show the pointer structure for each of the element types. A pointer element simply points to another element in ETAB. A set element points to a pointer list which in turn points to a set of elements in ETAB. Graph and subgraph elements point to an array of three pointers which point to pointer lists designating the node-set, branch set, and subgraph set of the graph (or subgraph). A node element contains a pointer to a pointer list designating the set of incident branches of the node. A branch element points to an array of two pointers which point to the tail and the head of the branch. Deletion of element is accomplished by changing the type field to "GB". Garbage collection is performed by periodically updating all sets, subgraphs, and graphs by deleting references to the elements of type "GB" and actually free the space occupied by these elements and their structural definitions. Since the sets are explicitly freed, when they are of no further use, our garbage collection routine is mainly concerned with cleaning up the graphs.

Since the list processing in Pℓ/1 is performed using based variables, we had to use these variables to perform all the list processing algorithms for SOL. Wherever possible it is recommended that the type of the elements involved in the search be specified to avoid unnecessary time consuming linear searches. We have used a unique entry structure for all different types of elements, which will leave some fields unused for certain types of elements.

The language was implemented using the top down parser generated by the TACOS compiler system. Actions are written in a Pℓ/1 program approximately 1000 statements long. SOL execution time routines are also written in Pℓ/1.

The SOL compiler generates PL/1 programs which simulate the actions intended by the input SOL program. These generated PL/1 programs are lengthy and PL/1 compiler is terribly slow. It is worthwhile to try an interpretive version of SOL, which is useful in an interactive SOL.

Sample SOL program

```
TRNS:  PROC (PTR1,BNAME);
   %/* THIS PROCEDURE WILL CREATE ALL THE BRANCHES IMPLIED BY THE
     TRANSITIVITY OF THE BRANCHES WITH LABEL "BNAME" IN THE GRAPH POINTED
     TO BY PTR1 */

     DCL  (PTR1,PTR2,B1,B2) POINTER,
          (BNAME,NAME,NAME1)CHAR(8); %
     FOR  (ALL,B,BOF(PTR1));
     NAME = NAME(B);
     IF   'NAME = BNAME'  THEN DO;
     PTR2 = HEAD(B);
     FOR  (ALL,B1,OUTBR(PTR2));
     NAME1 = NAME(131);
     IF   'NAME1 = BNAME'  THEN
     ADD  BR  BNAME(TAIL(B1),HEAD(B1))  TO PTR1;
     END;END;
     END;
END TRNS; .
```

Syntax used in current implementation

```
Sol-program:: = {PL1-statement|sol-statements}*

PL1-statements:: = %{PL1-statement}* %

sol-statement:: = labels{block|declaration|association|operation|if-statement
                  |else-statement|assignment|go-to-statement|end-statement
                  |call-statement};

block:: = BEGIN|{PROCEDURE|PROC}[(<*I>{,<*I>}*]|DO

labels:: = {<*I>:}*

declaration:: = {DECLARE|DCL}[integer]<*I>attribute
                {,[integer]<*I>attribute}*

attribute:: = {GRAPH|GR}[UNORIENTED]|
              {SUBGRAPH|SG}[DISJOINT]|
              {NODE|ND}|
              {BRANCH|BR}({<*I>|node-ref},{<*I>|node-ref})|
              {SET|ST}|
              {POINTER|PT}

association:: = associate|free

associate:: = {PTASOC|STASOC|NDASOC|BRASOC|SGASOC|GRASOC}
              [(element-ref{,element-ref}*)]PL1-declaration-tail;

PL1-declaration-tail:: = PL1-element-dcl{,PL1-element-dcl}*

free:: = {PTFREE|STFREE|NDFREE|BRFREE|SGFREE|GRFREE}
         [(element-ref{,element-ref}*)]<*I>{,<*I>}*

set-ref:: = sets-set|node-set|graph-set|[SET|ST]<*I>|pointer-ref

sets-set:: = set-set-mnemonic (set-ref,set-ref)

set-set-mnemonic:: = UNION|INTER|DIF|SYMDIF

node-ref:: = branch-node|[NODE|ND] qualified-name|pointer-ref

branch-node:: = {HEAD|TAIL}(branch-ref)
```

```
branch-ref:: = [BRANCH|BR]⟨*I⟩(node-ref,node-ref)
               [BRANCH|BR]qualified-name|pointer-ref
qualified-name:: = ⟨*I⟩{.⟨*I⟩}*
node-set:: = {ADJBR|INCBR|OUTBR}(node-ref)
graph-ref:: = [GRAPH|GR]⟨*I⟩|pointer-ref
pointer-ref:: = [POINTER|PT]⟨*I⟩
graph-set:: = {NOF|BOF}({graph-ref|subgraph-ref})
subgraph-ref:: = [SUBGRAPH|SG]qualified-name[DISJOINT]|pointer-ref
operation:: = loop-control|data-operation|pointer-operation|node-operation
              |subgraph-operation
element-operation:: = sets-set|sets-boolean|graph-set|node-set|branch-node
                      |set-integer|element-string
sets-boolean:: = {EQUALS|SUBSET}(set-ref,set-ref)
set-integer:: = CARD(set-ref)
element-string:: = {NAME|TYPE}(element-ref)
element-ref:: = pointer-ref|set-ref|node-ref|branch-ref
                |subgraph-ref|graph-ref
loop-control:: = FOR({ANY|ALL|integer},⟨*I⟩,set-ref)
data-operation:: = add-operation|delete-operation
add-operation:: = ADD element-ref TO element-ref
delete-operation:: = {DELETE|DEL}element-ref FROM element-ref
pointer-operation:: = {JUMP|MOVE|OMOVE}pointer-ref[{branch-ref|node-ref}
                      |label-name]|{MOVE2|OMOVE2}pointer-ref
                      [{branch-ref|label-name}[{node-ref|label-name}]
                      |{node-ref|label-name}[{branch-ref|label-name}]]
label-name:: = {LABEL|L}⟨*I⟩
```

```
node-operation:: = {PARTITION|GENERATE}node-ref[INTO]
                   {graph-ref|subgraph-ref} AS⟨*I⟩;
subgraph-operation:: = BACKUP node-ref|{MERGE|PARSE|DISCONNECT|SINK|
                       SOURCE|LINK}set-ref
                       [{label-name|branch-ref}][AS⟨*I⟩];
*go to-statement:: = {GO TO|GOTO}⟨*I⟩
 end-statement:: = END [⟨*I⟩]
*call-statement:: = CALL ⟨*I⟩[(⟨*I⟩,{,⟨*I⟩}*)]
*if-statement:: = IF character-string THEN if-tail[else-statement]
*if-tail:: = {Pℓ1-statement|sol-statement}*
*else-statement:: = ELSE if-tail
assignment:: = set-assignment|equ-assignment|assoc-assignment|nor-assignment
set-assignment:: = set-ref = (element-ref{,element-ref}*)
equ-assignment:: = #⟨*I⟩ = qualified-name
assoc-assignment:: = qualified-name (element-ref)
                     [(Pℓ1-index{,Pℓ1-index}*)] =
                     {qualified-name(element-ref)[(Pℓ1-index
                     {,Pℓ1-index}*)]|Pℓ1-expression}
nor-assignment:: = ⟨*I⟩[(Pℓ1-index{,Pℓ1-index}*)] =
                   {element-operation|qualified-name(element-ref)
                   [(Pℓ1-index{,Pℓ1-index}*)]|type qualified-name|
                   Pℓ1-expression}
type:: = {POINTER|PT}|{SET|ST}|{NODE|ND}|{BRANCH|BR}|{SUBGRAPH|SG}|{GRAPH|GR}
```

Note: All the phrases whose name starts with Pℓ1 are defined as their counterpart in Pℓ/1.

⟨*I⟩ is an identifier, and is defined as the identifiers in Pℓ/1. We allow maximum of 8 characters for each identifier.

Integer is a decimal whole number.

Character-string is a string of any combination of all permissible characters (including double quotation mark) between two quotation marks.

Reserved Words

ADD, ADJBR, ALL, ANY, AS

BEGIN, BETWEEN, BOF, BR, BRANCH, BRASOC, BRFREE

CALL, CARD

DCL, DECLARE, DEL, DELETE, DIF, DISJOINT, DISCONNECT, DO

ELSE, END, EQUALS

FOR, FROM

GENERATE, GO, GO TO, GR, GRAPH, GRASOC, GRFREE

HEAD

IF, INCBR, INTER, INTO

JUMP

L, LABEL, LINK

MERGE, MOVE, MOVE2

NAME, ND, NDASOC, NDFREE, NODE, NOF

OMOVE, OMOVE2, OUTBR

PARSE, PROC, PROCEDURE, PT, PTASOC, PTFREE, POINTER, PARTITION

SET, SG, SGASOC, SGFREE, SINK, SOURCE, ST, STASOC, STFREE, SUBSET, SYMDIF

TAIL, THEN, TO, TYPE

UNION, UNORIENTED

APPENDIX B

ADDITIONAL EXAMPLES

Here we have included additional examples of scene analysis. Fig. B.1 shows a rather complex scene. Fig. B.2 is its graphical representation. In the following pages, the SOL program which created this graph and its associations is given, and it is followed by the computer output of the results of this analysis.

Fig. B.3(a) displays a scene (chair) whose regions are divided, by occlusion or additional structure, into several sub-regions. A T-joint merging procedure will merge these regions and conform the input to our model graph. Then the chair is recognized normally. The analysis results are shown in the following pages.

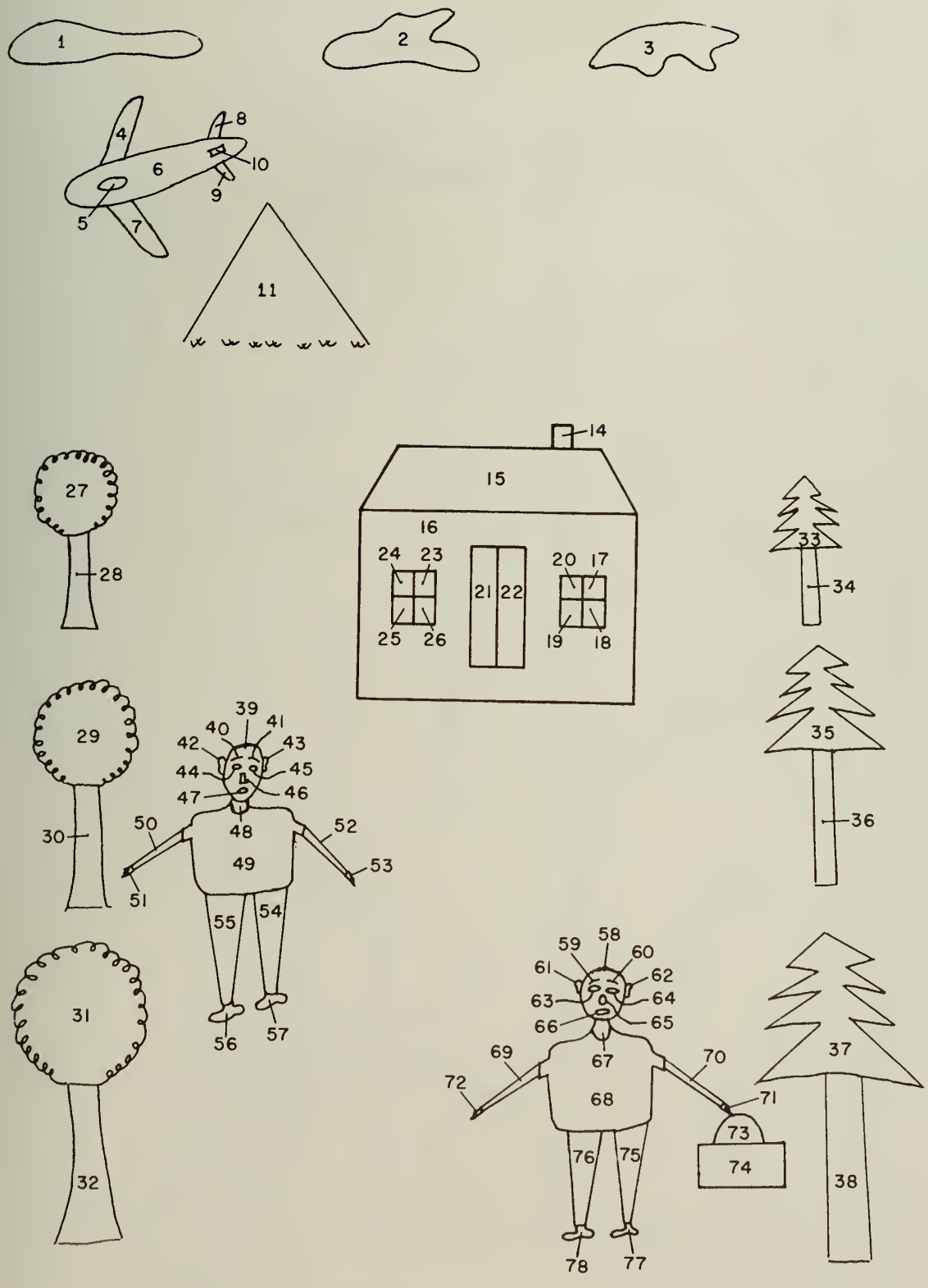


Fig. B.1. A more complex scene example.

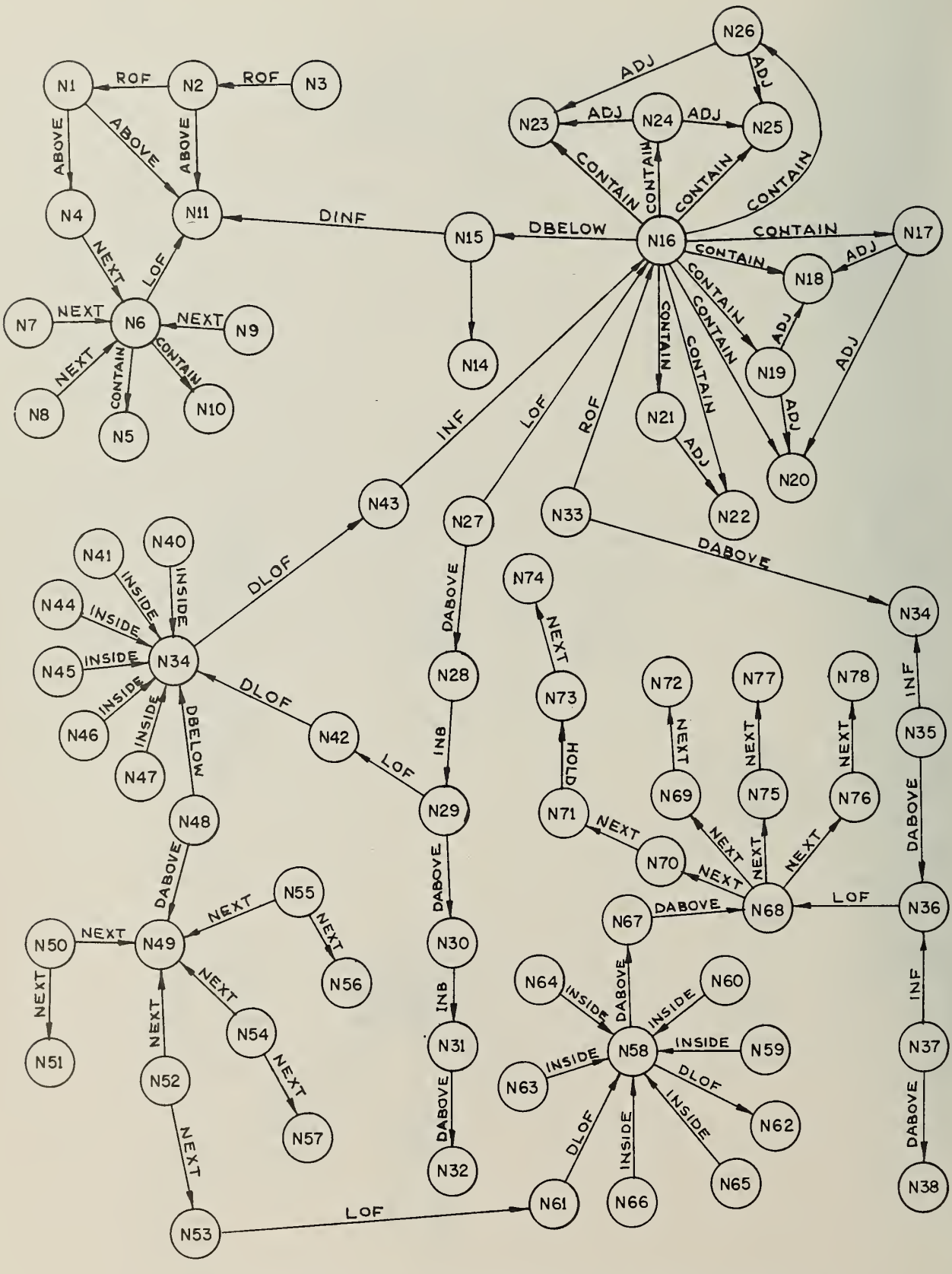


Fig. B.2. Graphical representation of scene in Fig. B.1.

```
   NONAME
   PARM = DISJOINT                                                    1
EXAM: PROC OPTIONS(MAIN) ;                                            2
%  /* THIS IS AN EXAMPLE OF A SCENE*/                                 3
   DCL GRAPH POINTER EXTERNAL,                                        4
   SMOOTH ENTRY(POINTER) ,                                            5
       SEE BIT(1) EXTERNAL ,                                          6
       SRCTRL POINTER CONTROLLED EXTERNAL ;                           7
   DCL RECOG ENTRY(DEC FIXED,DEC FIXED) ;                             8
   ALLOCATE SRCTRL ; SRCTRL=NULL ;                                    9
   ALLOCATE SRCTRL ;  %                                              10
   DCL PIC9 GR ,                                                     11
   ABOVE BR (N1,N4) ,                                                12
   ROF BR(N2,N1) ,                                                   13
   ABOVE BR(N1,N11),                                                 14
   ABOVE BR (N2,N11) ,                                               15
   ROF BR(N3,N2) ,                                                   16
   LOF BR(N6,N11) ,                                                  17
   CONTAIN BR (N6,N5) ,                                              18
   CONTAIN BR (N6,N10) ,                                             19
   NEXT BR(N4,N6) ,                                                  20
   NEXT BR(N7,N6) ,                                                  21
   NEXT BR(N8,N6) ,                                                  22
   NEXT BR(N9,N6) ,                                                  23
   INF BR(N15,N11) ,                                                 24
   DBELOW BR(N15,N14) ,                                              25
   DABOVE BR(N15,N16) ,                                              26
   ADJ BR(N24,N23) ,                                                 27
   ADJ BR(N24,N25) ,                                                 28
   ADJ BR(N26,N25) ,                                                 29
   ADJ BR(N26,N23) ,                                                 30
   ADJ BR(N21,N22) ,                                                 31
   ADJ BR(N17,N18) ,                                                 32
   ADJ BR(N17,N20) ,                                                 33
   ADJ BR(N19,N18) ,                                                 34
   ADJ BR(N19,N20) ,                                                 35
   CONTAIN BR (N16,N17) ,                                            36
   CONTAIN BR (N16,N18) ,                                            37
   CONTAIN BR (N16,N19) ,                                            38
   CONTAIN BR (N16,N20) ,                                            39
   CONTAIN BR (N16,N21) ,                                            40
   CONTAIN BR (N16,N22) ,                                            41
   CONTAIN BR (N16,N23) ,                                            42
   CONTAIN BR (N16,N24) ,                                            43
   CONTAIN BR (N16,N25) ,                                            44
   CONTAIN BR (N16,N26) ,                                            45
   LOF BR (N27,N16) ,                                                46
   ROF BR (N33,N16) ,                                                47
   INB BR(N28,N29) ,                                                 48
   INB BR(N30,N31) ,                                                 49
   LOF BR (N29,N42) ,                                                50
   INF BR (N35,N34) ,                                                51
   INF BR (N37,N36) ,                                                52
   LOF BR (N71,N37) ,                                                53
   INF BR (N43,N16) ,                                                54
   DABOVE BR (N27,N28) ,                                             55
   DABOVE BR (N29,N30) ,                                             56
                                                                     57
```

```
DABOVE BR (N31,N32) ,                                                  58
DABOVE BR(N33,N34) ,                                                   59
DABOVE BR (N35,N36) ,                                                  60
DABOVE BR (N37,N38) ,                                                  61
DLOF BR (N42,N39) ,                                                    62
DROF BR (N43,N39) ,                                                    63
INSIDE BR (N40,N39) ,                                                  64
INSIDE BR (N41,N39) ,                                                  65
INSIDE BR (N44,N39) ,                                                  66
INSIDE BR (N45,N39) ,                                                  67
INSIDE BR (N46,N39) ,                                                  68
INSIDE BR (N47,N39) ,                                                  69
DBELOW BR (N48,N39) ,                                                  70
DABOVE BR (N48,N49) ,                                                  71
NEXT BR (N50,N49) ,                                                    72
NEXT BR (N52,N49) ,                                                    73
NEXT BR (N54,N49) ,                                                    74
NEXT BR (N55,N49) ,                                                    75
NEXT BR (N50,N51) ,                                                    76
NEXT BR (N52,N53) ,                                                    77
NEXT BR (N55,N56) ,                                                    78
NEXT BR (N54,N57) ,                                                    79
ROF BR (N61,N53) ,                                                     80
DLOF BR (N61,N58) ,                                                    81
DLOF BR (N58,N62) ,                                                    82
INSIDE BR (N59,N58) ,                                                  83
INSIDE BR (N60,N58) ,                                                  84
INSIDE BR (N63,N58) ,                                                  85
INSIDE BR (N64,N58) ,                                                  86
INSIDE BR (N65,N58) ,                                                  87
INSIDE BR (N66,N58) ,                                                  88
DABOVE BR (N58,N67) ,                                                  89
DABOVE BR (N67,N68) ,                                                  90
NEXT BR (N68,N69) ,                                                    91
NEXT BR (N68,N70) ,                                                    92
NEXT BR (N68,N75) ,                                                    93
NEXT BR (N68,N76) ,                                                    94
NEXT BR (N69,N72) ,                                                    95
NEXT BR (N70,N71) ,                                                    96
NEXT BR (N76,N78) ,                                                    97
NEXT BR (N75,N77) ,                                                    98
HOLD BR (N71,N73) ,                                                    99
NEXT BR (N73,N74) ;                                                   100
NDASOC (NOF(PIC9)) 1 SHAPE , 2 SHP DEC FIXED,                         101
2 LTOW DEC FIXED (5,2) ;                                              102
SHAPE.SHP (PIC9.N1 )=22;                                              103
SHAPE.SHP (PIC9.N2 )=22;                                              104
SHAPE.SHP (PIC9.N3 )=22;                                              105
SHAPE.SHP (PIC9.N4 )=3 ;                                              106
SHAPE.SHP (PIC9.N5 )=4 ;                                              107
SHAPE.SHP (PIC9.N6 )=13;                                              108
SHAPE.SHP (PIC9.N7 )=3 ;                                              109
SHAPE.SHP (PIC9.N8 )=3 ;                                              110
SHAPE.SHP (PIC9.N9 )=3 ;                                              111
SHAPE.SHP (PIC9.N10)=1 ;                                              112
SHAPE.SHP (PIC9.N11)=28;                                              113
SHAPE.SHP (PIC9.N14)=1 ;                                              114
```

```
SHAPE.SHP (PIC9.N15)=8 ;                                    115
SHAPE.SHP (PIC9.N16)=1 ;                                    116
SHAPE.SHP (PIC9.N17)=1 ;                                    117
SHAPE.SHP (PIC9.N18)=1 ;                                    118
SHAPE.SHP (PIC9.N19)=1 ;                                    119
SHAPE.SHP (PIC9.N20)=1 ;                                    120
SHAPE.SHP (PIC9.N21)=1 ;                                    121
SHAPE.SHP (PIC9.N22)=1 ;                                    122
SHAPE.SHP (PIC9.N23)=1 ;                                    123
SHAPE.SHP (PIC9.N24)=1 ;                                    124
SHAPE.SHP (PIC9.N25)=1 ;                                    125
SHAPE.SHP (PIC9.N26)=1 ;                                    126
SHAPE.SHP (PIC9.N27)=18;                                    127
SHAPE.SHP (PIC9.N28)=1 ;                                    128
SHAPE.SHP (PIC9.N29)=18;                                    129
SHAPE.SHP (PIC9.N30)=1 ;                                    130
SHAPE.SHP (PIC9.N31)=18;                                    131
SHAPE.SHP (PIC9.N32)=1 ;                                    132
SHAPE.SHP (PIC9.N33)=19;                                    133
SHAPE.SHP (PIC9.N34)=8 ;                                    134
SHAPE.SHP (PIC9.N35)=19;                                    135
SHAPE.SHP (PIC9.N36)=8 ;                                    136
SHAPE.SHP (PIC9.N37)=19;                                    137
SHAPE.SHP (PIC9.N38)=8 ;                                    138
SHAPE.SHP (PIC9.N39)=4 ;                                    139
SHAPE.SHP (PIC9.N40)=27;                                    140
SHAPE.SHP (PIC9.N41)=27;                                    141
SHAPE.SHP (PIC9.N42)=24;                                    142
SHAPE.SHP (PIC9.N43)=24;                                    143
SHAPE.SHP (PIC9.N44)=4 ;                                    144
SHAPE.SHP (PIC9.N45)=4 ;                                    145
SHAPE.SHP (PIC9.N46)=13;                                    146
SHAPE.SHP (PIC9.N47)=8 ;                                    147
SHAPE.SHP (PIC9.N48)=29;                                    148
SHAPE.SHP (PIC9.N49)=7 ;                                    149
SHAPE.SHP (PIC9.N50)=8 ;                                    150
SHAPE.SHP (PIC9.N51)=23;                                    151
SHAPE.SHP (PIC9.N52)=8 ;                                    152
SHAPE.SHP (PIC9.N53)=23;                                    153
SHAPE.SHP (PIC9.N54)=8 ;                                    154
SHAPE.SHP (PIC9.N55)=8 ;                                    155
SHAPE.SHP (PIC9.N56)=9 ;                                    156
SHAPE.SHP (PIC9.N57)=9 ;                                    157
SHAPE.SHP (PIC9.N58)=4 ;                                    158
SHAPE.SHP (PIC9.N59)=27;                                    159
SHAPE.SHP (PIC9.N60)=27;                                    160
SHAPE.SHP (PIC9.N61)=24;                                    161
SHAPE.SHP (PIC9.N62)=24;                                    162
SHAPE.SHP (PIC9.N63)=4 ;                                    163
SHAPE.SHP (PIC9.N64)=4 ;                                    164
SHAPE.SHP (PIC9.N65)=13;                                    165
SHAPE.SHP (PIC9.N66)=8 ;                                    166
SHAPE.SHP (PIC9.N67)=29;                                    167
SHAPE.SHP (PIC9.N68)=7 ;                                    168
SHAPE.SHP (PIC9.N69)=8 ;                                    169
SHAPE.SHP (PIC9.N70)=8 ;                                    170
SHAPE.SHP (PIC9.N71)=23;                                    171
```

```
SHAPE.SHP (PIC9.N72)=23;                                    172
SHAPE.SHP (PIC9.N73)=6 ;                                    173
SHAPE.SHP (PIC9.N74)=1 ;                                    174
SHAPE.SHP (PIC9.N75)=8 ;                                    175
SHAPE.SHP (PIC9.N76)=8 ;                                    176
SHAPE.SHP (PIC9.N77)=9 ;                                    177
SHAPE.SHP (PIC9.N78)=9 ;                                    178
SHAPE.LTOW(PIC9.N1 )=5.6 ;                                  179
SHAPE.LTOW(PIC9.N2 )=6.7 ;                                  180
SHAPE.LTOW(PIC9.N3 )=4.3 ;                                  181
SHAPE.LTOW(PIC9.N4 )=3.5 ;                                  182
SHAPE.LTOW(PIC9.N5 )=2.1 ;                                  183
SHAPE.LTOW(PIC9.N6 )=7.3 ;                                  184
SHAPE.LTOW(PIC9.N7 )=3.5 ;                                  185
SHAPE.LTOW(PIC9.N8 )=2.3 ;                                  186
SHAPE.LTOW(PIC9.N9 )=2.3 ;                                  187
SHAPE.LTOW(PIC9.N10)=1.8 ;                                  188
SHAPE.LTOW(PIC9.N11)=1.7 ;                                  189
SHAPE.LTOW(PIC9.N14)=1.4 ;                                  190
SHAPE.LTOW(PIC9.N15)=4.5 ;                                  191
SHAPE.LTOW(PIC9.N16)=1.4 ;                                  192
SHAPE.LTOW(PIC9.N17)=1.1 ;                                  193
SHAPE.LTOW(PIC9.N18)=1.1 ;                                  194
SHAPE.LTOW(PIC9.N19)=1.1 ;                                  195
SHAPE.LTOW(PIC9.N20)=1.1 ;                                  196
SHAPE.LTOW(PIC9.N21)=4.5 ;                                  197
SHAPE.LTOW(PIC9.N22)=4.5 ;                                  198
SHAPE.LTOW(PIC9.N23)=1.2 ;                                  199
SHAPE.LTOW(PIC9.N24)=1.2 ;                                  200
SHAPE.LTOW(PIC9.N25)=1.2 ;                                  201
SHAPE.LTOW(PIC9.N26)=1.2 ;                                  202
SHAPE.LTOW(PIC9.N27)=1.2 ;                                  203
SHAPE.LTOW(PIC9.N28)=7.3 ;                                  204
SHAPE.LTOW(PIC9.N29)=1.2 ;                                  205
SHAPE.LTOW(PIC9.N30)=7.3 ;                                  206
SHAPE.LTOW(PIC9.N31)=1.2 ;                                  207
SHAPE.LTOW(PIC9.N32)=7.3 ;                                  208
SHAPE.LTOW(PIC9.N33)=4.7 ;                                  209
SHAPE.LTOW(PIC9.N34)=5.2 ;                                  210
SHAPE.LTOW(PIC9.N35)=4.7 ;                                  211
SHAPE.LTOW(PIC9.N36)=5.2 ;                                  212
SHAPE.LTOW(PIC9.N37)=4.7 ;                                  213
SHAPE.LTOW(PIC9.N38)=5.2 ;                                  214
SHAPE.LTOW(PIC9.N39)=1.2 ;                                  215
SHAPE.LTOW(PIC9.N40)=4.3 ;                                  216
SHAPE.LTOW(PIC9.N41)=4.3 ;                                  217
SHAPE.LTOW(PIC9.N42)=3.5 ;                                  218
SHAPE.LTOW(PIC9.N43)=3.5 ;                                  219
SHAPE.LTOW(PIC9.N44)=1.5 ;                                  220
SHAPE.LTOW(PIC9.N45)=1.5 ;                                  221
SHAPE.LTOW(PIC9.N46)=3.3 ;                                  222
SHAPE.LTOW(PIC9.N47)=1.8 ;                                  223
SHAPE.LTOW(PIC9.N48)=1.4 ;                                  224
SHAPE.LTOW(PIC9.N49)=1.3 ;                                  225
SHAPE.LTOW(PIC9.N50)=7.6 ;                                  226
SHAPE.LTOW(PIC9.N51)=1.7 ;                                  227
SHAPE.LTOW(PIC9.N52)=7.6 ;                                  228
```

```
SHAPE.LTOW(PIC9.N53)=1.7 ;                                    229
SHAPE.LTOW(PIC9.N54)=5.4 ;                                    230
SHAPE.LTOW(PIC9.N55)=5.4 ;                                    231
SHAPE.LTOW(PIC9.N56)=3.1 ;                                    232
SHAPE.LTOW(PIC9.N57)=3.1 ;                                    233
SHAPE.LTOW(PIC9.N58)=1.2 ;                                    234
SHAPE.LTOW(PIC9.N59)=4.3 ;                                    235
SHAPE.LTOW(PIC9.N60)=4.3 ;                                    236
SHAPE.LTOW(PIC9.N61)=3.5 ;                                    237
SHAPE.LTOW(PIC9.N62)=3.5 ;                                    238
SHAPE.LTOW(PIC9.N63)=1.5 ;                                    239
SHAPE.LTOW(PIC9.N64)=1.5 ;                                    240
SHAPE.LTOW(PIC9.N65)=3.2 ;                                    241
SHAPE.LTOW(PIC9.N66)=1.8 ;                                    242
SHAPE.LTOW(PIC9.N67)=1.4 ;                                    243
SHAPE.LTOW(PIC9.N68)=1.3 ;                                    244
SHAPE.LTOW(PIC9.N69)=7.5 ;                                    245
SHAPE.LTOW(PIC9.N70)=7.5 ;                                    246
SHAPE.LTOW(PIC9.N71)=1.7 ;                                    247
SHAPE.LTOW(PIC9.N72)=1.7 ;                                    248
SHAPE.LTOW(PIC9.N73)=11.3 ;                                   249
SHAPE.LTOW(PIC9.N74)=1.8 ;                                    250
SHAPE.LTOW(PIC9.N75)=5.4 ;                                    251
SHAPE.LTOW(PIC9.N76)=5.4 ;                                    252
SHAPE.LTOW(PIC9.N77)=3.1 ;                                    253
SHAPE.LTOW(PIC9.N78)=3.1 ;                                    254
SRCTRL=PIC9.N15;                                              255
/* GRAPH=PIC9 ;                                               256
CALL SMOOTH(GRAPH) ;                                          257
SEE='0'B ;                                                    258
CALL RECOG(0,0) ;     /*                                      259
END;                                                          260
```

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. MOUNTAIN
2. CLOUD
3. CLOUD
4. PLANE
5. HOUSE
6. CLOUD
7. TREE
8. TREE
9. MAN
10. TREE
11. TREE
12. MAN
13. TREE
14. TREE
15. BAG

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *RAG**.

REGION *N73** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N37** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N31** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *MAN**.

OBJECT *FACE**.

REGION *N61** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N35** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N29** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *MAN**.

OBJECT *FACE**.

REGION *N43** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N23** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *TREE**.

REGION *N27** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *N3** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *HOUSE**.

OBJECT *ROOF**.

REGION *N15** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *PLANE**.

REGION *N6** IS THE ATTENTICN POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *N2** IS THE ATTENTICN POINT OF PARSING THE ABOVE STEPS.

OBJECT *CLOUD**.

REGION *N1** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

OBJECT *MOUNTAIN**.

REGION *N11** IS THE ATTENTICN POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

A *MOUNTAIN** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *MOUNTAIN** IS LOCATED BELOW THE *CLOUD**.
THIS *MOUNTAIN** IS LOCATED BELOW THE SECOND *CLOUD**.
THIS *MOUNTAIN** IS LOCATED AT THE RIGHT OF THE *PLANE**.
THIS *MOUNTAIN** IS LOCATED IN BEHIND OF THE *HOUSE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED ABOVE THE *MOUNTAIN**.
THIS *CLOUD** IS LOCATED AT THE LEFT OF THE SECOND *CLOUD**.
THIS *CLOUD** IS LOCATED ABOVE THE *PLANE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED ABOVE THE *MOUNTAIN**.
THIS *CLOUD** IS LOCATED AT THE RIGHT OF THE *CLOUD**.
THIS *CLOUD** IS LOCATED AT THE LEFT OF THE THIRD *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *PLANE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *PLANE** IS LOCATED BELOW THE *CLOUD**.
THIS *PLANE** IS LOCATED AT THE LEFT OF THE *MOUNTAIN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *HOUSE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *HOUSE** IS LOCATED IN FRONT OF THE *MOUNTAIN**.
THIS *HOUSE** IS LOCATED AT THE RIGHT OF THE *TREE**.
THIS *HOUSE** IS LOCATED AT THE LEFT OF THE SECOND *TREE**.
THIS *HOUSE** IS LOCATED IN BEHIND OF THE *MAN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *CLOUD** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *CLOUD** IS LOCATED AT THE RIGHT OF THE SECOND *CLOUD**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *TREE** IS LOCATED AT THE LEFT OF THE *HOUSE**.
THIS *TREE** IS LOCATED IN BEHIND OF THE THIRD *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *TREE** IS LOCATED AT THE RIGHT OF THE *HOUSE**.
THIS *TREE** IS LOCATED IN BEHIND OF THE FOURTH *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A *MAN** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *MAN** IS LOCATED IN FRONT OF THE *HOUSE**.
THIS *MAN** IS LOCATED AT THE RIGHT OF THE THIRD *TREE**.
THIS *MAN** IS LOCATED AT THE LEFT OF THE SECOND *MAN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A THIRD *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *TREE** IS LOCATED IN FRONT OF THE *TREE**.
THIS *TREE** IS LOCATED AT THE LEFT OF THE *MAN**.
THIS *TREE** IS LOCATED IN BEHIND OF THE FIFTH *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A FOURTH *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:
THIS *TREE** IS LOCATED IN FRONT OF THE SECOND *TREE**.
THIS *TREE** IS LOCATED IN BEHIND OF THE SIXTH *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SECOND *MAN** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *MAN** IS LOCATED AT THE RIGHT OF THE *MAN**.

THIS *MAN** IS LOCATED AT THE LEFT OF THE SIXTH *TREE**.

THE RELATION *HOLD** BETWEEN THE *MAN** AND THE *BAG
IS NOT KNOWN TO THE RECOGNIZER.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A FIFTH *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *TREE** IS LOCATED IN FRONT OF THE THIRD *TREE**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

A SIXTH *TREE** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

THIS *TREE** IS LOCATED IN FRONT OF THE FOURTH *TREE**.

THIS *TREE** IS LOCATED AT THE RIGHT OF THE SECOND *MAN**.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

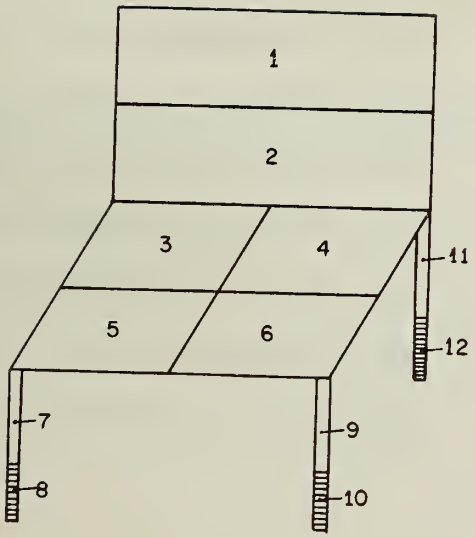
A *BAG** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

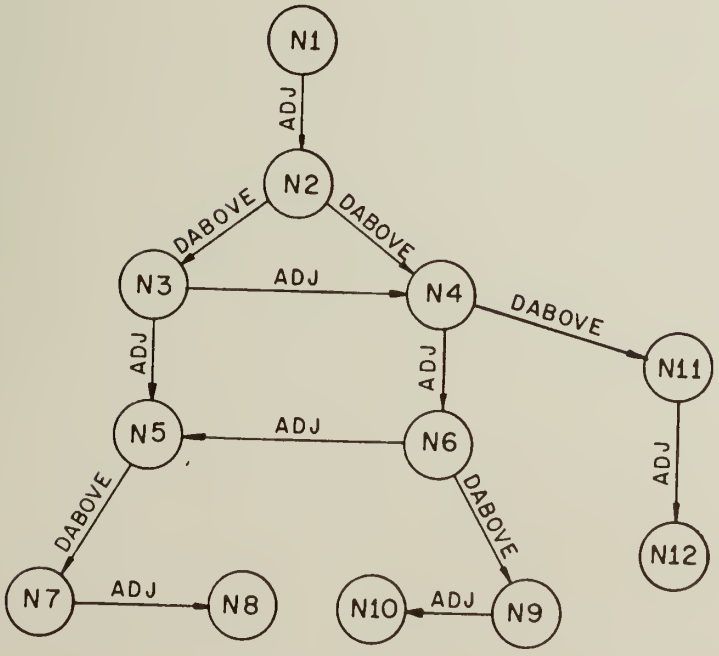
THE RELATION *HOLD** BETWEEN THE *BAG** AND THE SECOND *MAN
IS NOT KNOWN TO THE RECOGNIZER.

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.



(a) picture



(b) input graph

Fig. B.3. A scene example with divided regions.

```
NODES N2        AND N1        ARE MERGED INTO ONE NODE,NAMED GEN11
NODES N4        AND N3        ARE MERGED INTO ONE NODE,NAMED GEN12
NODES N5        AND N6        ARE MERGED INTO ONE NODE,NAMED GEN13
NODES N8        AND N7        ARE MERGED INTO ONE NODE,NAMED GEN14
NODES N10       AND N9        ARE MERGED INTO ONE NODE,NAMED GEN15
NODES N12       AND N11       ARE MERGED INTO ONE NODE,NAMED GEN16
NODES GEN12     AND GEN13     ARE MERGED INTO ONE NODE,NAMED GEN17
```

BRIEF DESCRIPTION OF THE SCENE.

THE FOLLOWING OBJECTS WERE PARSED IN THE SCENE.

1. CHAIR

END OF BRIEF DESCRIPTION.

FULL DESCRIPTION OF THE SCENE.

THE FOLLOWING SUCCESSFUL STEPS WERE TAKEN IN PARSING THE SCENE.

OBJECT *CHAIR**.

REGION *GEN11** IS THE ATTENTION POINT OF PARSING THE ABOVE STEPS.

THIS CONCLUDES THE REPORT OF THE SUCCESSFUL ATTEMPTS.

THE PARSED PICTORIAL INFORMATION IS AS FOLLOWS:

A *CHAIR** IS FOUND IN THE SCENE.

ITS RELATIONS TO THE OTHER PARTS OF THE SCENE ARE AS FOLLOWS:

END OF RELATIONAL DESCRIPTION FOR THIS OBJECT.

END OF SCENE DESCRIPTION.

VITA

Ahmad Eftekhari Masumi was born in Teheran, Iran, on December 11, 1944. He was accepted as a foreign student under the Japanese Government scholarship to Japan and graduated from the University of Tokyo, Japan, with a Bachelor of Engineering in Electronics in 1968. Early in 1970 he finished his graduate studies at the University of Tokyo and received a masters degree in Electrical Engineering. He continued his research at the same university until later in 1970.

From 1970 to 1973 he was a research assistant in the Department of Computer Science at the University of Illinois.

Mr. Masumi is a student member of the Association for Computing Machinery. He is also a professional member of the Institute for Electronics and Communication Engineers of Japan, and Information Processing Society of Japan.

Form AEC–427
(6/68)
AECM 3201

U. S. ATOMIC ENERGY COMMISSION

UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

C00-2118-0049

2. TITLE

Picture Analysis by Graph Transformation

3. TYPE OF DOCUMENT (Check one):

[X] a. Scientific and technical report

[ ] b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

[ ] c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

[X] a. AEC's normal announcement and distribution procedures may be followed.

[ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.

[ ] c. Make no announcement or distrubution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

Ahmad E. Masumi
Research Assistant

Organization

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Signature

Ahmad E. Masumi

Date

October 1973

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

[ ] a. AEC patent clearance has been granted by responsible AEC patent group.

[ ] b. Report has been sent to responsible AEC patent group for clearance.

[ ] c. Patent clearance not required.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-73-604	2.	3. Recipient's Accession No.
4. Title and Subtitle Picture Analysis by Graph Transformation			5. Report Date October 1973 6.
7. Author(s) Ahmad E. Masumi			8. Performing Organization Rept. No. UIUCDCS-R-73-604
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No. US AEC AT(11-1)2118 11. Contract/Grant No. 46-26-15-303
12. Sponsoring Organization Name and Address US AEC Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois 60439			13. Type of Report & Period Covered 14.
15. Supplementary Notes			

16. Abstracts

In this thesis, we have developed a methodology to analyze the pictorial information represented in a map-like image, and further name the objects most likely present in the scene based on the universe known to the system.

In contrast to conventional string language sentences which are formed by concatenating the primitives (symbols) in a string according to the language grammar, the pictorial sentences are basically formed of primitives which are located in a two-dimensional (or three-dimensional) space with much richer relational properties than simple adjacency. Here we have examined the possibility of describing the structural information of pictures as syntactical rules, employing a rich class of relations. The graphs are known to be extremely convenient and flexible to represent this kind of information. By expressing the syntactical properties of one class of pictures as a collection of graphs, we have shown that algorithms can be easily developed to parse objects or collections of objects which belong to this class. Further, this methodology is flexible enough to recognize incomplete objects, and has learning capabilities through adding and modifying the rules of the system. In addition, we have shown that a complete description of an object as a collection of graphs can be easily modified (rotated) to enable the system to recognize different projections of the object. It has also been encouraging to note that heuristical information is easily introducible to the system and immensely improves the performance of the system.

17. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Release	UNCLASSIFIED	213
	20. Security Class (This Page) UNCLASSIFIED	22. Price -----